Title
Adaptive/Optimal Vehicle Infrastructure Integration With Intelligent Vehicles and Environmentally-Friendly Continuous Flow Network Design

Author
Kari, David

Publication Date
2016

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Adaptive/Optimal Vehicle Infrastructure Integration With Intelligent Vehicles and Environmentally-Friendly Continuous Flow Network Design

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

David Prem Kari

August 2016

Dissertation Committee:
Dr. Matthew J. Barth, Chairperson
Dr. Qi Zhu
Dr. Kanok Boriboonsomsin
Dr. David R. Cocker

The Dissertation of David Prem Kari is approved:

Committee Chairperson

University of California, Riverside

Acknowledgements

The author is indebted to several individuals and organizations for their encouragement and support. First, the author would like to thank each of the members of the dissertation committee including Dr. Matthew Barth, Dr. Qi Zhu, Dr. David Cocker, and Dr. Kanok Boriboonsomsin, for their inspiration, encouragement, and interest over the past several years. In particular, the head of the dissertation committee, the author's graduate advisor Dr. Barth, provided both employment as a member of the Transportation Systems Research group at the Bourns College of Engineering Center for Environmental Research and Technology (CE-CERT) at the University of California at Riverside (UCR), as well as the inspiration for much of the material presented in the following dissertation. In addition, the author is also grateful for the countless hours of discussion and collaboration with Dr. Guoyuan Wu and Dr. Qiu ("Apple") Jin. Dr. Wu is also thanked for his development of one of the Paramics traffic networks used in the dissertation. Finally, the author would like to note that portions of the work presented in the following dissertation were funded by projects including the "Eco-Lanes Operational Scenario" portion of the AERIS-Applications for the Environment: Real-Time Information Synthesis project supported by the Intelligent Transportation Systems Joint Project Office (ITS JPO) at the U.S. Department of Transportation (USDOT), and the "Using Connected Vehicle Technology for Advanced Signal Control Strategies" and "Eco-Friendly Intelligent Transportation System Technology for Freight Vehicles"

projects funded through the National Center for Sustainable Transportation (NCST) with the aid of the California Energy Commission (CEC).

ABSTRACT OF THE DISSERTATION

Adaptive/Optimal Vehicle Infrastructure Integration With Intelligent Vehicles and Environmentally-Friendly Continuous Flow Network Design

by

David Prem Kari

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, August 2016
Dr. Matthew J. Barth, Chairperson

The American transportation system faces unprecedented challenges today and will face even more significant challenges in the near future. According to Texas A&M's 2012 Urban Mobility Report, the combined effects of wasting 5.5 billion hours and 2.9 billion gallons of fuel annually, resulted in a total congestion cost of $121 billion in the United States. In addition, according to the National Highway Traffic Safety Administration (NHTSA), approximately 33,000 traffic fatalities occurred in 2014 with an additional estimated 53,000 annual deaths due to transportation-related emissions. Each of these challenges is compounded by the prospect of the U.S. population increasing 44% by 2050. The field of Intelligent Transportation Systems (ITS) can play a major role in meeting the aforementioned challenges and in creating a 21st century transportation system that is safer, more reliable, more efficient, and more sustainable than the existing transportation system, while remaining affordable. A total of three solutions are proposed in this dissertation with varying ability to improve the mobility

and environmental sustainability of traffic. The first solution is the application of Eco-Cooperative Adaptive Cruise Control (Eco-CACC) and the associated lane-changing algorithms with Connected Vehicles (CVs) to freeway traffic systems. The second solution is the leveraging of CV technology to provide real-time adaptive signal control for arterial intersections. Finally, the third solution is the development of an eco-friendly city designed around a proposed novel continuous flow intersection.

Table of Contents

Acknowledgements ..... **iii**

ABSTRACT OF THE DISSERTATION ..... **v**

1. Introduction **1**

2. Background **9**

2.1 Connected Vehicles 9

2.2 Cooperative Adaptive Cruise Control 10

2.3 Conventional Traffic Signal Control 13

2.4 Intersection Design 15

2.4.1 Introduction to UAIDs 15

2.4.1.1 Motivation and Rationale for New Intersection Designs 15

2.4.1.2 Description of Performance Metrics for Intersections 17

2.4.2 Description of Existing Intersection Designs 22

2.4.2.1 The Double-T Intersection 23

2.4.2.2 The Near-side Jughandle Intersection 24

2.4.2.3 The Far-side Jughandle Intersection 25

2.4.2.4 The Median U-Turn Intersection (MUT) 27

2.4.2.5 The Restricted Crossing Median U-Turn Intersection (RCUT) 29

2.4.2.6 The Super-street Median Crossover Intersection (SSM) 30

2.4.2.7 The Quadrant Roadway Intersection (QR) 31

2.4.2.8 The Split Intersection 32

2.4.2.9 The Bowtie Median Intersection 33

2.4.2.10 The Parallel Flow Intersection (PFI) 34

2.4.2.11 The Displaced Left-Turn Intersection (DLT/CFI) 37

2.4.2.12 The Double-Crossover Intersection (DXI) 39

2.4.2.13 The Upstream Signalized Crossover Intersection (USC) 40

2.4.2.14 The Round-about 41

2.4.2.15 The Hamburger Intersection 42

2.4.3 Comparison of UAIDs 43

3. Software Design **55**

4. Vehicle Enhancement Techniques **60**

4.1 Eco-Cooperative Adaptive Cruise Control (Eco-CACC) 60
4.1.1 Application Description 61
4.1.2 Algorithm Design 62
4.1.3 Cooperative Maneuver Protocols 70
4.1.4 Modeling Approach 76
4.1.5 Eco-CACC Results 77
4.1.5.1 Simulation Networks Description 77
4.1.5.1.1 Network #1: Hypothetical Freeway Segment with Lane-Drop 77
4.1.5.1.2 Network #2: Hypothetical Freeway Segment with On/Off Ramps & a Dedicated Lane 78
4.1.5.1.3 Network #3: California SR-91 Eastbound Freeway with Dedicated Lane 79
4.1.5.2 Network 1: Hypothetical Freeway Segment with Lane-Drop 81
4.1.5.2.1 Traffic Volume Sensitivity Analysis 81
4.1.5.2.2 Eco-CACC Penetration Rate Sensitivity Analysis 86
4.1.5.2.3 Triggering Distance Sensitivity Analysis 89
4.1.5.2.4 Intra-Platoon Spacing Sensitivity Analysis 92
4.1.5.3 Network 2: Hypothetical Freeway Segment with On-Ramps/Off-Ramps and a Dedicated Lane (“Eco-Lane”) 95
4.1.5.3.1 Traffic Volume Sensitivity Analysis 96
4.1.5.3.2 Traffic Network Savings vs. Dedicated Lane Savings 100
4.1.5.3.3 Benefits of Mainstream Vehicles Choosing the Dedicated Lane 103
4.1.5.4 Network 3: SR-91 E Freeway with Dedicated Lane (“Eco-Lane”) 106
4.1.5.4.1 Traffic Volume Sensitivity Analysis 107
4.1.5.4.2 Traffic Network Savings vs. Dedicated Lane Savings 116
4.1.5.4.3 Benefits for Mainstream Vehicles Choosing the Dedicated Lane 120
4.2 The Role of Acceleration and Deceleration in Eco-tuning of Eco-Cooperative Adaptive Cruise Control 122
5. Integrating Infrastructure Enhancement Techniques with Connected Vehicles 129
5.1 Adaptive/Optimal Signal Control with Intelligent Vehicles 129
5.1.1 Isolated Intersection Connected Vehicle Signal Optimization 129
5.1.1.1 Application Description 129
5.1.1.1.1 System Introduction 130

5.1.1.1.2 System Architecture 130
5.1.1.1.3 Fixed & Flexible Traffic Light State Machines 133
5.1.1.1.4 Signal Timing Optimization and MOE Selection 139
5.1.1.2 Simulation Setup 145
5.1.1.3 PARAMICS Network Description 146
5.1.1.4 Volume Sensitivity Analysis 148
5.1.1.4.1 Connected Vehicle Queue Length Signal Optimization versus Fixed Phase Signal Timing 149
5.1.1.4.2 Connected Vehicle Queue Length Signal Optimization versus Webster Signal Timing 154
5.1.1.5 Demand Profile Sensitivity Analysis 159
5.1.2 Corridor-Level Connected Vehicle Signal Optimization 164
5.1.2.1 Application Description 164
5.1.2.2 Simulation Setup 164
5.1.2.3 PARAMICS Network Description 165
5.1.2.4 Volume Sensitivity Analysis 168
6. Eco-Friendly Infrastructure Re-design for Intelligent Vehicles 189
6.1 Derivation of Continuous Flow Intersections & Networks 189
6.1.1 Description of the Euclidean Steiner Tree Problem and the Minimum Spanning Tree Problem 189
6.1.2 Tiling of the 4-Point Euclidean Steiner Tree 192
6.1.3 The Hexagonal Tree as an Urban Transportation Network 193
6.1.3.1 Optimality of the Hexagonal Tiling 193
6.1.3.2 Operational Modalities for Hexagonal Traffic Networks 194
6.1.3.2.1 3-Phase Signal Control of a Hexagonal Traffic Network 195
6.1.3.2.2 Signal-less Control of a Hexagonal Traffic Network 200
6.1.4 Introduction of Signal-less 3-way Intersections as a Basis for Continuous Flow Networks 202
6.1.4.1 Introduction of Known 3-vertex Euclidean Tilings 203
6.1.5 Additional Continuous Flow Networks 209
6.1.5.1 The Rounded Rectified Square Network 209
6.1.5.2 The Martini Lattice Network 213

6.1.6 Network Comparisons 216
6.1.6.1 Network Operations Comparison 216
6.1.6.2 Circuity Comparison 217
6.1.6.3 Land Utilization Comparison 220
6.2 Proposed Continuous Flow Intersection Results 225
6.2.1 Simulation Setup 225
6.2.2 PARAMICS Networks Description 227
6.2.2.1 Proposed UAID PARAMICS Networks 227
6.2.2.1.1 Rounded Truncated Square Intersection 227
6.2.2.2 Baseline PARAMICS Networks 230
6.2.2.2.1 Conventional 4-leg Intersection 230
6.2.2.2.2 Round-about Intersection 232
6.2.2.2.3 Parallel Flow Intersection (PFI) 234
6.2.3 Volume Sensitivity Analysis 238
6.2.3.1 Proposed Design vs. Conventional 4-leg Intersection 239
6.2.3.2 Proposed Design vs. Round-about Intersection 244
6.2.3.3 Proposed Design vs. Parallel Flow Intersection 249
6.2.3.4 Proposed Design vs. all 3 Baselines 255
6.2.4 Additional Sensitivity Analyses 259
6.2.4.1 Major/Minor Ratio Sensitivity Analysis 259
6.2.4.2 Demand Profile Sensitivity Analysis 262
6.2.4.3 Left-turn Percentage Sensitivity Analysis 264
6.2.5 Range of Benefits 268
6.2.6 Benefit-Cost Analysis 269
6.2.7 Design Parameter Optimization for Truncated Square Intersection 275
7. Conclusions & Future Work 283
8. References 304
Appendix A: Candidates for Eco-CACC Controllers 309
Appendix B: Freeway CACC Code 314
Appendix C: Isolated Intersection Connected Vehicle Signal Optimization Code 370
Appendix D: Corridor-Level Connected Vehicle Signal Optimization Code 419

List of Tables

Table 1: Motor vehicle Level-Of-Service (LOS) thresholds at signalized intersections 19
Table 2: Geometric and Operational Comparison of UAIDs..... 45
Table 3: Conflict Point Comparison of Existing UAIDs 46
Table 4: Conflict Point Rank Ordering of UAIDs 47
Table 5: Origin-Demand (OD) Matrix Route Conflict Points for UAIDs 49
Table 6: Total OD Route Conflict Point Rank Ordering of UAIDs 50
Table 7: Origin-Demand (OD) Matrix Route Crossing Points for UAIDs 52
Table 8: Total OD Route Conflict Point Rank Ordering of UAIDs 53
Table 9: Space-Time Categorization of Traffic Modelling..... 56
Table 10: ITS Eco-CACC Definitions 62
Table 11: Baseline, Traffic Volume Sensitivity Analysis Results..... 85
Table 12: Eco-CACC, Traffic Volume Sensitivity Analysis Results 85
Table 13: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results..... 85
Table 14: Travel Time % Improvement of Eco-CACC over Baseline, Penetration Rate Sensitivity Analysis Results..... 88
Table 15: Energy % Improvement of Eco-CACC over Baseline, Penetration Rate Sensitivity Analysis Results..... 89
Table 16: Travel Time % Improvement of Eco-CACC over Baseline, Triggering Distance Sensitivity Analysis Results..... 91
Table 17: Energy % Improvement of Eco-CACC over Baseline, Triggering Distance Sensitivity Analysis Results..... 92
Table 18: Travel Time % Improvement of Eco-CACC over Baseline, Intra-Platoon Spacing Sensitivity Analysis Results..... 94
Table 19: Energy % Improvement of Eco-CACC over Baseline, Intra-Platoon Spacing Sensitivity Analysis Results..... 95
Table 20: Baseline, Traffic Volume Sensitivity Analysis Results..... 99
Table 21: Eco-CACC, Traffic Volume Sensitivity Analysis Results 99
Table 22: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results..... 100
Table 23: Baseline, Traffic Volume Sensitivity Analysis Results..... 111
Table 24: Eco-CACC, Traffic Volume Sensitivity Analysis Results, 5m Vehicle Spacing..... 112
Table 25: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results, 5m Vehicle Spacing 112
Table 26: Baseline, Traffic Volume Sensitivity Analysis Results..... 115
Table 27: Eco-CACC, Traffic Volume Sensitivity Analysis Results, 15m Vehicle Spacing..... 115
Table 28: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results, 15m Vehicle Spacing 116

Table 29: Travel Time (seconds) Results for Eco-CACC Acceleration Test 127
Table 30: Energy (kJ/mi) Results for Eco-CACC Acceleration Test 127
Table 31: Travel Time (seconds) Results for Eco-CACC Deceleration Test 127
Table 32: Energy (kJ/mi) Results for Eco-CACC Deceleration Test 128
Table 33: Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results 153
Table 34: CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results..... 153
Table 35: % Improvement of CV Queue Length Optimization over Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results..... 154
Table 36: Webster Signal Timing, Traffic Volume Sensitivity Analysis Results 158
Table 37: CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results..... 159
Table 38: % Improvement of CV Queue Length Optimization over Webster Signal Timing, Traffic Volume Sensitivity Analysis Results..... 159
Table 39: Demand Profile Sequences used for Demand Profile Sensitivity Test..... 161
Table 40: Percent Travel Time Sensitivity to Varied Demand Profile 162
Table 41: Travel Time Sensitivity to Varied Demand Profile 163
Table 42: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results..... 174
Table 43: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results..... 174
Table 44: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results 175
Table 45: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results for Coordinated Phase Vehicles..... 179
Table 46: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results for Coordinated Phase Vehicles 180
Table 47: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results, for Coordinated Phase Vehicles 180
Table 48: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results for Uncoordinated Phase Vehicles..... 185
Table 49: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results for Uncoordinated Phase Vehicles 186
Table 50: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results, for Uncoordinated Phase Vehicles 186
Table 51: Operational Modalities for 3-way Intersections 195
Table 52: Comparison of Network Operational Modalities and Geometric Features..... 217
Table 53: Route Distance Comparison of Continuous Flow Networks vs. Conventional Grid Network 219

Table 54: Maximum Percent Delay of Conventional Network under which a Conventional Network will outperform a Continuous Flow Network in terms of Travel Time 220
Table 55: Packing Factor Percentage for Continuous Flow Networks with Largest Cell Area of 1 million square feet..... 225
Table 56: Packing Factor Percentage for Continuous Flow Networks with Average Cell Area of 1 million square feet..... 225
Table 57: Conventional 4-leg Intersection, Traffic Volume Sensitivity Analysis Results 243
Table 58: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results..... 244
Table 59: % Improvement of Truncated Square Intersection over Conventional 4-leg Intersection, Traffic Volume Sensitivity Analysis Results..... 244
Table 60: Round-about Intersection, Traffic Volume Sensitivity Analysis Results 248
Table 61: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results..... 249
Table 62: % Improvement of Truncated Square Intersection over Round-about Intersection, Traffic Volume Sensitivity Analysis Results..... 249
Table 63: PFI, Traffic Volume Sensitivity Analysis Results..... 253
Table 64: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results..... 254
Table 65: % Improvement of Truncated Square Intersection over PFI, Traffic Volume Sensitivity Analysis Results..... 254
Table 66: Major/Minor Ratio Sensitivity Results for Travel Time..... 262
Table 67: Demand Profile Sequences used for Demand Profile Sensitivity Test 263
Table 68: Demand Profile Sensitivity Results for Travel Time..... 264
Table 69: Left-turn Sensitivity Results for Travel Time..... 267
Table 70: Range of Benefits for Truncated Square Intersection over Baseline Intersections..... 269
Tables 71-73: Financial Cost Inputs of BCA..... 270
Table 74: Financial Cost of Conventional 4-leg Intersection and a Truncated Square Intersection..... 271
Table 75: Benefit-Cost Analysis Inputs 272
Table 76: Simulated and Projected Values for Benefit-Cost Analysis 274
Table 77: Benefit-Cost Analysis Equations 275
Table 78: Benefit-Cost Analysis..... 275
Table A.1 Eco-CACC Controller Options 309

List of Figures

Figure 1: Primary Traffic Objectives Categories, *Financial Sustainability is treated as a specifiable constraint 2
Figure 2: Sources of Greenhouse Gas Emissions [EPA, 2011] 4
Figure 3: Parallel View of Traffic System Layers 6
Figure 4: Pyramid View of Traffic System Layers 7
Figure 5: NEMA Dual-Ring Phasing Diagram [“Signal Timing on a Shoestring”, 2005] 14
Figure 6: NEMA Dual-Ring Phasing Diagram [“Signal Timing on a Shoestring”, 2005] 16
Figure 7: Conventional Intersection Vehicular Conflict Points [Rodegerdts *et al*., 2004] 20
Figure 8: Conventional Intersection Pedestrian Conflict Points [Robinson, *et*. *al*., 2000] 20
Figure 9: Conventional Intersection Bicyclist Conflict Points [Chandler, *et*. *al*., 2013] 21
Figure 10: Double-T Intersection with Conflict Points Overlay [Rodegerdts *et al*., 2004] 24
Figure 11: Near-side Jughandle Intersection with Conflict Points Overlay [Rodegerdts *et al*., 2004] 25
Figure 12: Far-side Jughandle Intersection [Rodegerdts *et al*., 2004] 27
Figure 13: Median U-Turn Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004] ... 29
Figure 14: Restricted Crossing Median U-Turn Intersection with Conflict Point Overlay [Esaway & Sayed, 2013] 30
Figure 15: Super-street Median Crossover Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004] 31
Figure 16: Quadrant Roadway Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004] 32
Figure 17: Split Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004] 33
Figure 18: Bowtie Median Intersection with Conflict Point Overlay [MUID, 2015] 34
Figure 19: Parallel Flow Intersection (Top) with Conflict Point Overlay (Bottom) [Parsons] 36
Figure 20: Displaced Left-Turn Intersection [Hughes *et al*., 2010] 38
Figure 21: Displaced Left-Turn Intersection Conflict Point Overlay [Rodegerdts *et al*., 2004] ... 39
Figure 22: Double-Crossover Intersection with Conflict Point Overlay [Esaway & Sayed, 2013] 40
Figure 23: Upstream Signalized Crossover Intersection with Conflict Point Overlay [Esaway & Sayed, 2013] 41
Figure 24: Round-about Intersection with Conflict Point Overlay [FHWA (“Safety Aspects of Roundabouts”)] 42
Figure 25: Hamburger Intersection with Conflict Point Overlay [Sangster *et al*., 2015] 43
Figure 26: Microscopic Traffic Simulation System Diagram 57
Figure 27: Illustration of Definitions 62
Figure 28: Freeway CACC Role State Machine 64
Figure 29: Freeway CACC State Flow Diagram 66
Figure 30: CACC “Follower” Level 2 Diagram, Spatial Regulator with Acceleration Constraints 69
Figure 31: CACC Splitting Protocol, Level 2 Diagram 73

Figure 32: CACC Lane Merging Protocol, Level 2 Diagram 75
Figure 33: CACC Merging Protocol; a) Prior to Stage 1, b) Completion of Stage 1, c) Completion of Stage 2 76
Figure 34: Aerial View of Lane-Drop Area of Network #1 in PARAMICS 78
Figure 35: Aerial View of Network #2 in PARAMICS 79
Figure 36: Aerial View of On/Off Ramp Pair in Network #2 in PARAMICS 79
Figure 37: Aerial View of Network #3 in PARAMICS 80
Figure 38: Aerial View of SR-91 / I-15 Interchange Model in PARAMICS 80
Figure 39: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates 83
Figure 40: Average Travel Time % Savings vs. Traffic Volume for 100% vs 0% Penetration Rates 83
Figure 41: Energy vs. Traffic Volume for 0% & 100% Penetration Rates 84
Figure 42: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates 84
Figure 43: Average Travel Time % Savings vs. Traffic Volume and Penetration Rate 87
Figure 44: Energy % Savings vs. Traffic Volume and Penetration Rate 88
Figure 45: Average Travel Time % Savings vs. Traffic Volume and Triggering Distance 90
Figure 46: Energy % Savings vs. Traffic Volume and Triggering Distance 91
Figure 47: Average Travel Time % Savings vs. Traffic Volume and Intra-Platoon Spacing 93
Figure 48: Energy % Savings vs. Traffic Volume and Intra-Platoon Spacing 94
Figure 49: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates 97
Figure 50: Average Travel Time % Savings vs. Traffic Volume for 100% vs 0% Penetration Rates 97
Figure 51: Energy vs. Traffic Volume for 0% & 100% Penetration Rates 98
Figure 52: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates 98
Figure 53: Dedicated Lane Energy Savings vs. Traffic Volume 102
Figure 54: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings 103
Figure 55: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane 105
Figure 56: Average Benefits for Mainstream Vehicles Starting in the Dedicated Lane 106
Figure 57: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates, 5m Vehicle Spacing 109
Figure 58: Average Travel Time % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 5m Vehicle Spacing 110
Figure 59: Energy vs. Traffic Volume for 0% & 100% Penetration Rates, 5m Vehicle Spacing 110
Figure 60: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 5m Vehicle Spacing 111
Figure 61: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates, 15m Vehicle Spacing 113
Figure 62: Average Travel Time % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 15m Vehicle Spacing 114
Figure 63: Energy vs. Traffic Volume for 0% & 100% Penetration Rates, 15m Vehicle Spacing 114

Figure 64: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 15m Vehicle Spacing 115
Figure 65: Dedicated Lane Energy Savings vs. Traffic Volume, 5m Vehicle Spacing 117
Figure 66: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings, 5m Vehicle Spacing 118
Figure 67: Dedicated Lane Energy Savings vs. Traffic Volume, 15m Vehicle Spacing 119
Figure 68: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings, 15m Vehicle Spacing 120
Figure 69: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane, 5m Vehicle Spacing 121
Figure 70: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane, 15m Vehicle Spacing 122
Figure 71: Travel Time Results for Eco-CACC Acceleration Test 125
Figure 72: Energy Results for Eco-CACC Acceleration Test 125
Figure 73: Travel Time Results for Eco-CACC Deceleration Test 126
Figure 74: Energy Results for Eco-CACC Deceleration Test 126
Figure 75: CV MAS Level 0 Diagram: Top-level System Architecture 131
Figure 76: CV MAS Level 1 Diagram: Intersection Management Agent Flow Chart 132
Figure 77: CV MAS Level 1 Diagram: Vehicle Agent Flow Chart 132
Figure 78: Signal Phase Diagram & Dual-Ring Controller, adapted from [“Signal Timing on a Shoestring”, 2005] 134
Figure 79: Fixed Sequence & Coupled Phase Dual-Ring Controller, Finite State Machine Representation 135
Figure 80: Flexible Dual-Ring Controller, Finite State Machine Representation 136
Figure 81: Main street portion of Flexible Dual-Ring Controller, Finite State Machine Representation 137
Figure 82: Side street portion of Flexible Dual-Ring Controller, Finite State Machine Representation 138
Figure 83: CV MAS Level 2 Diagram: Queue Length Signal Optimizer 142
Figure 84: Plot of Quadratic Aging Factor Equation 145
Figure 85: Isolated Intersection PARAMICS Network 147
Figure 86: Turning Movements for Isolated Intersection 148
Figure 87: Average Travel Time Comparison of CV Queue Length Signal Optimization and Fixed Phase Signal Timing for an Isolated Intersection 150
Figure 88: Average Travel Time Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection 151
Figure 89: Average Energy Comparison Of CV Queue Length Signal Optimization and Fixed Phase Signal Timing for an Isolated Intersection 151
Figure 90: Average Energy Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection 152

Figure 91: Average Emissions Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection 152
Figure 92: Average Travel Time Comparison of CV Queue Length Signal Optimization and Webster Signal Timing for an Isolated Intersection 156
Figure 93: Average Travel Time Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection 156
Figure 94: Average Energy Comparison of CV Queue Length Signal Optimization and Webster Signal Timing for an Isolated Intersection 157
Figure 95: Average Energy Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection 157
Figure 96: Average Emissions Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection 158
Figure 97: Percent Travel Time Sensitivity to Varied Demand Profile 162
Figure 98: Travel Time Sensitivity to Varied Demand Profile 163
Figure 99: 3-Intersection Corridor PARAMICS Network 167
Figure 100: Time-Space Diagram showing Coordinated Fixed Phase Signal Timing Plan for PARAMICS Network (see previous Figure) 167
Figure 101: Average Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 171
Figure 102: Average Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 172
Figure 103: Average Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 172
Figure 104: Average Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 173
Figure 105: Average Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 173
Figure 106: Coordinated Phase Vehicle Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 175
Figure 107: Coordinated Phase Vehicle Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 176
Figure 108: Coordinated Phase Vehicle Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 177
Figure 109: Coordinated Phase Vehicle Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 178
Figure 110: Coordinated Phase Vehicle Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 179

Figure 111: Uncoordinated Phase Vehicle Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 181
Figure 112: Uncoordinated Phase Vehicle Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 182
Figure 113: Uncoordinated Phase Vehicle Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor 183
Figure 114: Uncoordinated Phase Vehicle Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 184
Figure 115: Uncoordinated Phase Vehicle Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 185
Figure 116: Comparison of Average Travel Time Percent Savings by Category for Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 187
Figure 117: Comparison of Average Energy Percent Savings by Category for Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor 188
Figure 118: a) A suboptimal solution of the Minimum Spanning Tree problem, b) A suboptimal solution to the Euclidean Steiner Tree problem, c) Optimal solution to the Euclidean Steiner Tree problem; adapted from [Derksen, 2007] 190
Figure 119: a) Optimal solution of the Minimum Spanning Tree problem, b) a sub-optimal solution to the Euclidean Steiner Tree problem, and c) optimal solution to the Euclidean Steiner Tree Problem for four points lying at the corners of a unit square; adapted from [Larson and Odoni, 1981] 191
Figure 120: Steiner Tree with equivalent segment lengths 192
Figure 121: Tiling of an equivalent segment length Steiner Tree 193
Figure 122: Signal Phase Diagram for a 3-way Intersection 195
Figure 123: Traffic Movements for a Hexagonal Cell 197
Figure 124: “Network” Phases for 3-phase Signalized Hexagonal Network, a) Phase 1, b) Phase 2, c) Phase 3 198
Figure 125: Composite View of “Network” Phases for 3-phase Signalized Hexagonal Network, a) Phase 1, b) Phase 2, c) Phase 3 199
Figure 126: Conflict Point Diagram for a Three-Leg Intersection 200
Figure 127: Example of a Restricted Left-Turn Continuous Flow 3-way Intersection 201
Figure 128: Network Diagram for Signal-less Control of a Hexagonal Network 202
Figure 129: Regular Tessellations of the Euclidean Plane, a) triangular tiling, b) square tiling, and c) hexagonal tiling, adapted from [Parks *et al*., 2007] 204

Figure 130: Semiregular Tessellations of the Euclidean Plane, from left to right, top row, a) snub hexagonal tiling, b) elongated triangular tiling, c) truncated square tiling, d) snub square tiling, bottom row, e) rhombitrihexagonal tiling, f) trihexagonal tiling, g) truncated trihexagonal tiling, and h) truncated hexagonal tiling, adapted from [Parks *et al*., 2007] 204
Figure 131: The 4 tilings with 3 regular polygons per vertex, a) hexagonal tiling, b) truncated square tiling, c) truncated hexagonal tiling, and d) truncated trihexagonal tiling, adapted from [Parks *et al*., 2007] and [Harris, 2000]..... 205
Figure 132: Network Diagram for Signal-less Control of a Truncated Square Network 207
Figure 133: Network Diagram for Signal-less Control of a Truncated Hexagonal Network 208
Figure 134: Network Diagrams for Signal-less Control of a Truncated Trihexagonal Network, a) Modality 1 (left), b) Modality 2 (right)..... 209
Figure 135: Example of a Restricted Through Continuous Flow 4-way Intersection 211
Figure 136: Network Diagram for Signal-less Control of a Rounded Rectified Square Network 212
Figure 137: Comparison of Network Through Movement for a) square network (left), and b) rounded rectified square network (right) 212
Figure 138: The Martini Lattice [Ziff and Scullard, 2006]..... 214
Figure 139: Network Diagram for Signal-less Control of a Martini lattice Network, adapted from [Ziff and Scullard, 2006]..... 215
Figure 140: Packing Factor Percentage for Continuous Flow Networks with Largest Cell Area of 1 million square feet..... 223
Figure 141: Packing Factor Percentage for Continuous Flow Networks with Average Cell Area of 1 million square feet..... 224
Figure 142: a) Truncated Square Traffic Network, b) Truncated Square UAID (in inset)..... 228
Figure 143: a) Rounded Truncated Square Design, b) West junction of rounded truncated square design (in inset)..... 228
Figure 144: Direction of Traffic Flow within a rounded truncated square intersection 229
Figure 145: Rounded Truncated Square Intersection with West Approach Traffic Movements, Top to Bottom: 1) Left-turn movement, 2) Through movement, & 3) Right-turn movement..... 230
Figure 146: PARAMICS Network for 3-lane Conventional 4-leg Intersection..... 231
Figure 147: Turning Movements for Conventional 4-leg Intersection 232
Figure 148: PARAMICS Network for 3-lane Round-about Intersection 233
Figure 149: Turning Movements for Round-about Intersection..... 234
Figure 150: PARAMICS Network for 2-lane PFI 236
Figure 151: Phase 1 Signals for PFI 237
Figure 152: Phase 2 Signals for PFI 237
Figure 153: Average Travel Time Comparison of a Truncated Square Intersection vs. a Conventional 4-leg Intersection..... 241
Figure 154: Average Travel Time Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection..... 241

Figure 155: Average Energy Comparison of a Truncated Square Intersection vs. a Conventional 4-leg Intersection 242
Figure 156: Average Energy Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection 242
Figure 157: Average Emissions Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection 243
Figure 158: Average Travel Time Comparison of a Truncated Square Intersection vs. a Round-about Intersection 246
Figure 159: Average Travel Time Savings of a Truncated Square Intersection over a Round-about Intersection 246
Figure 160: Average Energy Comparison of a Truncated Square Intersection vs. a Round-about Intersection 247
Figure 161: Average Energy Savings of a Truncated Square Intersection over a Round-about Intersection 247
Figure 162: Average Emissions Savings of a Truncated Square Intersection over a Round-about Intersection 248
Figure 163: Average Travel Time Comparison of a Truncated Square Intersection vs. a PFI 251
Figure 164: Average Travel Time Savings of a Truncated Square Intersection over a PFI 251
Figure 165: Average Energy Comparison of a Truncated Square Intersection vs. a PFI 252
Figure 166: Average Energy Savings of a Truncated Square Intersection over a PFI 252
Figure 167: Average Emissions Savings of a Truncated Square Intersection over a PFI 253
Figure 168: Average Travel Time Comparison of Truncated Square Intersection and Baseline Intersections 257
Figure 169: Average Energy Comparison of Truncated Square Intersection and Baseline Intersections 258
Figure 170: Average Delay Time Comparison with LOS shown at Right 259
Figure 171: Major/Minor Ratio Sensitivity Results for Travel Time 261
Figure 172: Demand Profile Sensitivity Results for Travel Time 264
Figure 173: Left-turn Sensitivity Results for Travel Time 267
Figure 174: Geometric Design Parameters for Truncated Square Intersection 276
Figure 175: Percent Increase in Free-flow Travel Time of a Truncated Square Intersection defined by (*a*, *b*) over a Conventional 4-leg Intersection 278
Figure 176: Average Travel Time Percent Savings of a Truncated Square Intersection defined by (*a*, *b*) over a Conventional 4-leg Intersection 281
Figure 177: Absolute Free-flow Travel Time of a Truncated Square Intersection defined by (*a*, *b*) 282
Figure A.1: Spatial Regulator 310
Figure A.2: Spatial Regulator with Acceleration Constraints 311
Figure A.3: Temporal Regulator 312
Figure A.4: Temporal Regulator with Spatial and Acceleration Constraints 313

1. Introduction

The American transportation system faces unprecedented challenges today and will face even more significant challenges in the near future. The overall problem of a deteriorating traffic system may be divided into the categories of mobility, safety, environmental sustainability, and financial sustainability. The challenges among the categories are not independent of each other, but rather interrelated (see Figure 1 below). Improving the performance of a single category may yield benefits in the remaining categories. As a result, multiple system-wide performance metrics may be achievable by focusing on a single sub-aspect of the overall problem. Conversely, poor performance in any one category may negatively impact other categories. For example, poor mobility in traffic systems may endanger safety, have detrimental effects on the environment, and result in unnecessary financial costs. According to Texas A&M's 2012 Urban Mobility Report [Schrank *et al.*, 2012], the combined effects of wasting 5.5 billion hours and 2.9 billion gallons of fuel annually, (partly due to traffic accidents), resulted in a total congestion cost of $121 billion in the United States.

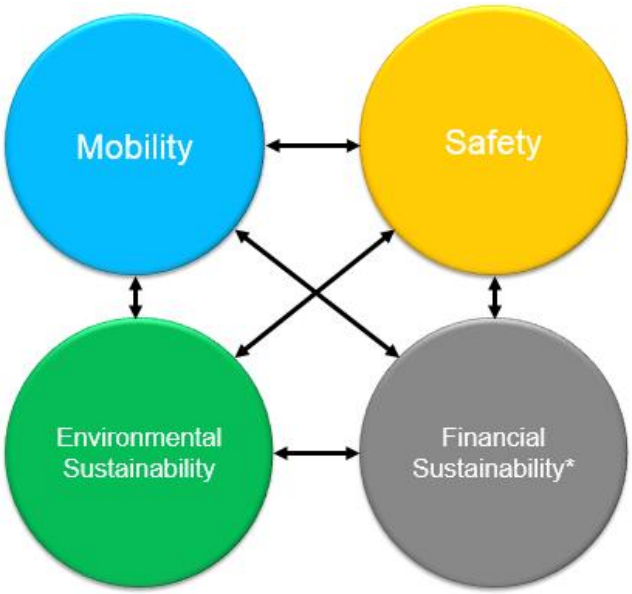

Figure 1: Primary Traffic Objectives Categories, *Financial Sustainability is treated as a specifiable constraint

Likewise, deficiencies in traffic safety can decrease mobility within a traffic network, and negatively impact environmental and financial sustainability. According to the latest National Highway Traffic Safety Administration (NHTSA) report, roughly 33,000 traffic fatalities occurred in 2014 with an additional 2.3 million injuries out of 6.1 million police-reported crashes [National Center for Statistics and Analysis, 2015] Traffic incidents not only increase the risk of secondary incidents near the initial crash site, but also impair system efficiency by blocking lanes. Temporary lane closures result in additional vehicles moving at reduced speed on accident free lanes; effectively creating a bottleneck.

The financial costs associated with a given traffic accident are far-reaching and include wage and productivity losses, medical expenses, administratrive expenses, vehicle and property damage, employer costs related to losing employee work hours, and lost quality of life for injured motorists, cyclists, or pedestrians. According to the

National Safety Council (NSC), the average comprehensive cost for an injury-free accident in 2014 was estimated to be $45,700. In addition, the average comprehensive cost for a single death due to a traffic accident was estimated to be $9,887,000 in 2014 [National Safety Council, 2016]. Coupling NHTSA fatality data and NSC estimations, the total comprehensive costs for traffic accidents in the USA occurring in 2014 was roughly $1 trillion.

The environmental impacts of traffic systems revolve around assessing the impact of greenhouse gas emissions. In 2011, the transportation sector contributed 28% of US greenhouse gas emissions [EPA, 2011], as shown in Figure 2 below. Unfortunately, several pollutants emitted by vehicles are known to be deadly to the human body. In fact, an estimated 53,000 people die every year due to transportation emissions [Fabio Caiazzo *et al*., 2013]. It is worth noting that the number of fatalities in the US due to emissions is roughly 60% greater than the number of fatalities due to traffic accidents. According to the available data, the environmental impact of traffic systems on urban populations poses an even more significant threat to human life than the actual safety record of the transportation system itself. In essence, the environmental impact of a given traffic system is an important component in a wholistic assessment of the overall safety of the traffic system, and cannot be ignored.

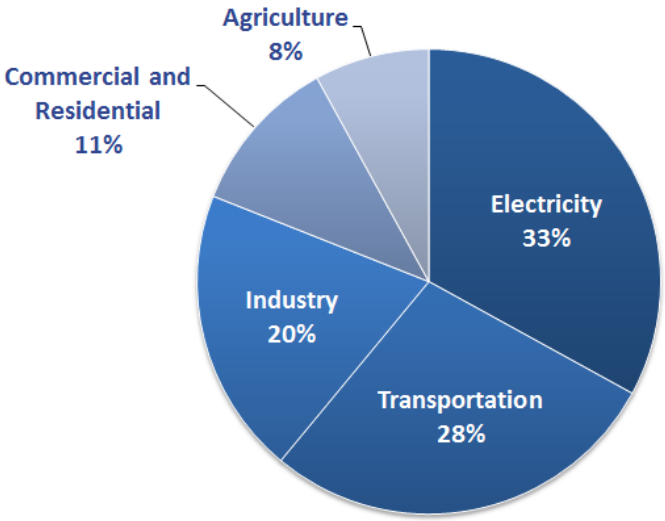

Figure 2: Sources of Greenhouse Gas Emissions [EPA, 2011]

In addition to the quantifiable mobility, safety, environmental, and financial costs of traffic accidents are the non-quantifiable costs associated with losing a single person to an accident. Every individual is unique and irreplacable, potentially possessing the ability and skill to radically alter the course of humanity. Even traffic engineers and urban planners are not immune to the risk of traffic accidents.

The challenges of traffic mobility, safety, environmental sustianibility, and financial cost are compounded by the prospect of the US population increasing 44% by 2050 [U.S. Census Bureau, 2008]. Based on all of the statistics mentioned, one must conclude that today's transportation system is dangerous, expensive, and unsustainable. If there are solutions to our transportation challenges, engineers have an ethical responsibility to pursue and develop them. A host of new and emerging technologies offer an unprecedented opportunity to re-think the way we transport goods and people, and to aim for a 21st century transportation system that is safe, reliable, affordable, and sustainable.

There are a wide range of potential solutions to creating a 21st century transportation system that is safer, more reliable, more efficient, and more sustainable while remaining affordable. The solutions may be classified based on their proposed treatment of vehicles, infrastructure, and the interaction or cooperation layer between vehicles and between vehicles and infrastructure. As shown in Figure 3 below, the three traffic system layers may be viewed as parallel. Vehicles may be modified (eg. electric vehicles, biofuels, etc.) or vehicles may be enhanced (eg. V2X and X2V communication). The term "modified" is used to denote alternative implementations that achieve similar or identical functionality. The term "enhanced" is used to indicate the addition of equipment to accommodate new capabilities. On the left side of the figure, infastructure may be enhanced to serve communicating vehicles (V2I). The development of the interaction/cooperation layer shown in the middle section of Figure 3, is critical for improving traffic performance for a given traffic network design. The state-of-the-practice interaction/cooperation layer relies on sensors such as inductive loop sensors, radar/LiDAR, and video cameras to sense traffic conditions. Each of the aforementioned sensors has drawbacks: loop detectors may fail to detect long queues at signalized intersections, radar/LiDAR-based systems retain classification error and may be troubled by occlusion, and video-based systems are not robust to light or weather conditions and require accurate tracking and data association. In contrast, Connected Vehicle (CV) communication technology is able to circumvent many of the sensor-based errors. As a result, the combination of developing infrastructure with communication technology and

vehicles with communication technology permits a greater level of cooperation between vehicles and infrastructure using more accurate and robust real-time data.

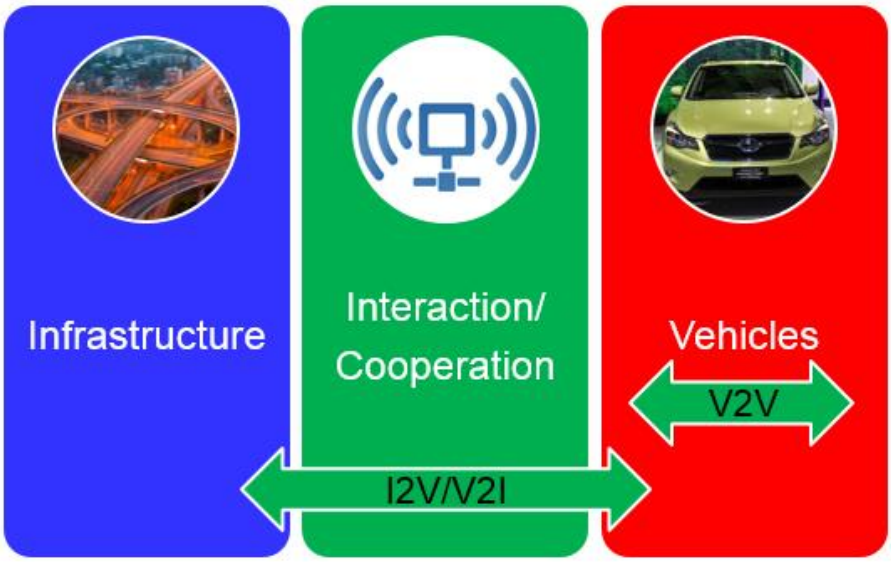

Figure 3: Parallel View of Traffic System Layers

In Figure 3 above, the three traffic layers were shown as parallel to indicate the importance of simultaneously enhancing infrastructure and vehicles to achieve an unprecedented level of cooperation. An alternative means of viewing the three traffic system layers is as a pyramid, as shown in Figure 4 below. Rather than simply enhancing existing infrastructure, infastructure may be re-designed to better accommodate vehicles and provide new opportunities for reducing the environmental footprint of traffic. Instead of optmizing around existing infrastructure that was not designed for connected and automated vehicles, a new infrastructure may be designed around 21st century needs.

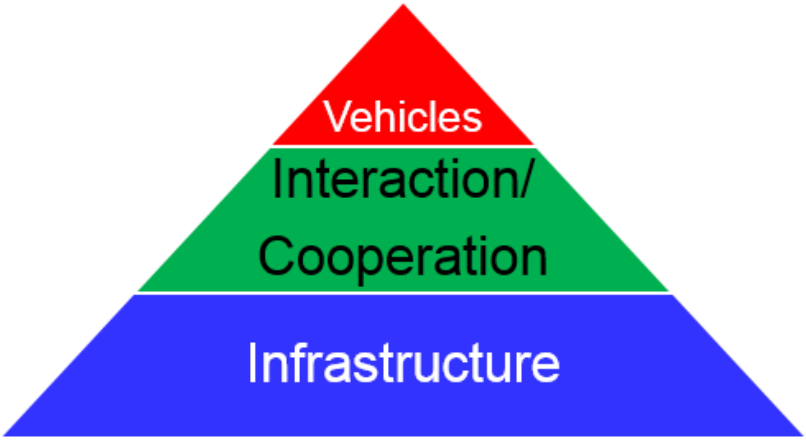

Figure 4: Pyramid View of Traffic System Layers

Using the two views of traffic system layers as a starting point, several different innovative solutions were developed, culminating in a number of key contributions. First, an environmentally-friendly approach towards improving freeway traffic with increases in freeway capacity was developed. In addition, the application of wireless communication between vehicles and traffic intersections was shown to be a viable solution for improving arterial traffic. The most significant contribution presented in the dissertation is a set of environmentally-friendly continuous flow traffic network designs which dramatically reduce vehicle emissions while cutting travel time.

The remainder of the dissertation is organized as follows. Chapter 2 includes background information on Connected Vehicles (CVs), Cooperative Adaptive Cruise Control (CACC), conventional traffic signal control as well as unconventional arterial intersection design (UAID). A description of the software architecture used for the micro-simulation platform is provided in chapter 3. Chapter 4 introduces an Eco-Cooperative Adaptive Cruise Control (Eco-CACC) system applied in the context of

designated lanes on freeways. In chapter 5, CV technology is applied to vehicles such that real-time vehicle information is used to dynamically control traffic signal timing. Chapter 6 presents several novel continuous flow traffic networks as well as the results for a newly designed continuous flow intersection. Finally, concluding remarks and future work is included in chapter 7.

2. Background

In the sections below, brief background material is provided on the key technical items addressed in this dissertation.

2.1 Connected Vehicles

The term "Connected Vehicles" (CVs) is used in the field of Intelligent Transportation Systems (ITS) to refer to vehicles which are equipped to communicate with and receive information from other vehicles and infrastructure. The "connection" portion of CV consists of the sharing and exchange of information. The communication involving CVs is categorized into several types including vehicle-to-vehicle communication (V2V), vehicle-to-infrastructure communication (V2I), and infrastructure-to-vehicle communication (I2V). Each of the categories is implemented using wireless communication. The two primary technologies most often considered for enabling wireless communication to and from vehicles are Dedicated Short Range Communication for Wireless Access in Vehicular Environments (DSRC/WAVE) radios and cellular communication devices. Referred to as DSRC for short, DSRC operates within a 75 MHz range in the 5.9 GHz band, as set forth by the Federal Communications Commission (FCC) in Report and Order FCC-03-324 ["Report and Order FCC-03-324," 2004]. DSRC has the advantage of using standardized message formats which enhance interoperability. Example message formats include the Basic Safety Message (BSM) (parts I and II), as standardized in the SAE J2735 standard [SAE J2735, 2009]. Additional advantages of DSRC include relatively low latency and high reliability

including under adverse weather conditions. One of the drawbacks of DSRC is its communication range, which is typically set to 300 meters, though ranges of up to 1000 meters are possible [Guo and Balon, 2006]. In contrast, cellular devices have a greater range, but are less reliable and are not designed specifically for vehicle safety applications. However, cellular devices can be used to augment DSRC with non-critical information such as traffic conditions 10 miles downstream.

Although DSRC was originally invented to support safety applications, both mobility and environmental applications can also benefit greatly from using DSRC. In fact, DSRC provides both the foundation and framework for nearly all ITS applications, and additional message formats are being designed specifically for further enabling ITS applications. The ITS applications presented in chapters 4 and 5 make use of DSRC within the context of V2V communications on freeways and V2I communications near traffic intersections.

2.2 Cooperative Adaptive Cruise Control

Due to the prospect of Dedicated Short Range Communications (DSRC) for Wireless Access in Vehicular Environments (WAVE) technologies being standardized on vehicles within the next few years, the introduction of relatively high penetration rates of Connected Vehicles (CVs) may be drawing near. Although the initial idea behind CVs is to improve vehicle occupant safety, a host of additional applications will be possible. One of the Intelligent Transportation Systems (ITS) concepts enabled by wireless communication is Cooperative Adaptive Cruise Control (CACC). The following

paragraphs highlight some of the key findings in the literature concerning CACC. A more detailed literature review concerning CACC was conducted, and may be found in [Bevly *et al.*, 2015].

At a minimum, a core CACC system functions like an Adaptive Cruise Control (ACC) system in that it provides longitudinal control to an equipped vehicle to maintain a safe distance from vehicles ahead. As an extension of an ACC system, however, a CACC system takes advantage of wireless communication by exchanging real-time information (e.g., vehicle's instantaneous acceleration) with surrounding vehicles (usually in the same platoon) and infrastructure using vehicle-to-vehicle (V2V) and infrastructure-to-vehicle (I2V) communications, respectively. Wireless vehicle communication may significantly improve efficiency and string stability of the system [Caudill and Garrard, 1977; Swaroop and Hedrick, 1996; Liang and Peng, 2000] by reducing the delay of the response to maneuvers of preceding vehicles [Lu *et al.*, 2002; van Arem *et al.*, 2006]. As demonstrated in a simulation study by Shladover *et al.*, the lane capacity was not significantly affected by ACC, but was nearly doubled with 100% penetration of CACC vehicles [Shaldover *et al.*, 2012]. Recent field studies on CACC systems [Bu *et al.*, 2010; Naus *et al.*, 2010; Milanés *et al.*, 2014] also validated their capabilities to improve the system throughput. Conventional ACC relies on sensors and actuators to detect and regulate the gap between a given vehicle and the vehicle directly ahead, whereas CACC adds communicated information from the vehicle directly ahead, and possibly the platoon leader, to permit shorter following gaps than are possible for ACC. For example, Nowakowski *et al.* reported that drivers were comfortable with a

minimum gap of 0.6 second (on average) when driving CACC-enabled vehicles, compared to a minimum gap setting of 1.1 seconds offered by factory-built ACC systems [Nowakowski *et al.*, 2010]. Aside from driver acceptance of following gaps, Öncü *et al.* examined the effects of wireless communication on CACC system performance [Öncü *et al.*, 2014]. Results indicated that communication limitations, such as network delay, influence the design of CACC controllers and the selected gap settings.

In addition to the core longitudinal control functionality, a number of studies in recent years have focused on lateral maneuvers for ACC or CACC systems, especially within weaving areas (e.g., at a highway ramp). Minimum longitudinal spacing and lateral acceleration trajectories have been well studied using vehicle dynamics models for the purpose of avoiding collisions while maintaining driver comfort during lane change maneuvers [Chee and Tomizuka, 1994; Jula *et al.*, 2000]. With wireless communication technology, CACC systems have significant potential to conduct lane change and merge maneuvers in a more cooperative manner. Xu and Sengupta simulated both ACC and CACC in the context of highway merging, and demonstrated that highway merging control with CACC could improve the efficiency of ramp merging with increasing average vehicle velocity and reducing braking effort [Xu and Sengupta, 2003]. Beyond the lane change of each individual vehicle, cooperative merging or splitting maneuvers on a platoon basis can also be fulfilled via different CACC protocols, depending on the traffic states on the merging or diverging lane [Lu *et al.*, 2004; Hsu and Liu, 2008].

While most of the existing studies have focused on the safety and mobility aspects of CACC systems, only a few have investigated the environmental benefits and they are primarily concerned with the benefits for CACC-equipped vehicles [Browand *et al.*, 2004; Tsugawa *et al.*, 2011]. The content presented in chapter 4 will seek to address these research gaps by extensively evaluating the system-wide environmental as well as mobility benefits of an Eco-CACC application. An Eco-CACC system is defined as a CACC system where the parameters have been tuned with special attention paid to environmental performance measures.

2.3 Conventional Traffic Signal Control

Conventional traffic signal control for 4-leg intersections uses the standard National Electrical Manufacturers Association (NEMA) signal phases, as shown in Figure 5 below [“Signal Timing on a Shoestring,” 2005]. For each leg of the intersection, there are three movements: a left-turn movement, a through movement, and a right-turn movement. Typically, the right-turn movement for a given intersection leg is permitted to be concurrent with the intersection leg’s through movement. Therefore, there are a total of eight signalized phases at a conventional 4-leg intersection. The eight phases are divided into main street and side street phases, as indicated in the right-side portion of Figure 5. The phases are further divided into 2 rings. Both rings, {1, 2, 3, 4}, and {5, 6, 7, 8}, consist of self-conflicting phases. Two phases are non-conflicting if they are on the same side of the barrier and in different rings. For example, phase 1 may be active with either phase 6 or phase 6. Each column shown in the phase table on the right-side

portion of Figure 5 represents a dual-ring signal phase. A typical cycle consists of serving the 8 individual phases with 4 dual-ring phases. The main street movements are usually served before the side street movements and are also given a larger "split" of the total cycle length time than the side street movements. In addition, the left-turn movements on each side of the barrier usually precede the through and right-turn movements.

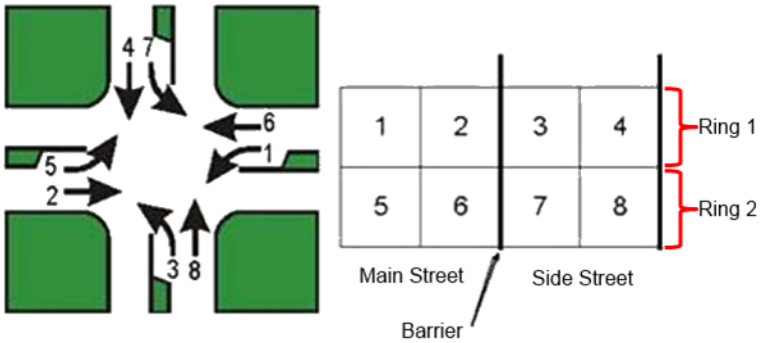

Figure 5: NEMA Dual-Ring Phasing Diagram ["Signal Timing on a Shoestring", 2005]

Traffic signals may be controlled based on either fixed signal timing or actuated/adaptive signal control timing. Fixed signal timing uses fixed cycle lengths and fixed signal splits based on historical traffic data and field observations. Actuated/adaptive signal control timing makes use of sensors such as inductive loop detectors (ILDs), video cameras, or radar/LiDAR sensors to modify signal timing based on the real-time arrivals of vehicles. The term "actuation" refers to the activation of one or more sensors, whereas "adaptive" is used to indicate that the signal timing is being modified based on the detection of vehicles. There are a number of adaptive signal

control optimization methods which have been deployed including OPAC, PRODYN, RHODES, SCAT, and SCOOT [Stevanovic, 2010].

2.4 Intersection Design

2.4.1 Introduction to UAIDs

2.4.1.1 Motivation and Rationale for New Intersection Designs

Traditional intersections require drivers to either stop at a stop-sign, wait at a traffic light, or to yield (as in a round-about). The current state of the art in arterial intersection design in the U.S. centers around 4-leg, 4-phase traffic signals, with a total of 8 signalized movements (Figure 1). As shown in Figure 1, an individual through and right-turn movement is considered as a single signalized movement. The through/right-turn movements account for 4 out of 8 of the signalized movements, with left turns accounting for the remaining 4 signalized movements. There are three general approaches to improving arterial intersection performance. First, vehicles can be enhanced to make optimal use of real-time traffic infrastructure information. For example, vehicles may receive Signal Phase and Timing (SPaT) messages from the nearest intersection to coordinate their approach to and departure from the signalized intersection. The aforementioned application is known as Eco-approach & departure (EAD) [Haitao *et al.*, 2013]. In addition, equipped vehicles may also coordinate their movement at the start of a green light to avoid queue discharge delay. Second, infrastructure may be enhanced to make optimal use of real-time vehicle state

information (i.e. location and speed). In the case of an arterial intersection, the SPaT can be modified based on wirelessly receiving the actual locations and speeds of equipped vehicles, instead of using existing sensor technology such as inductive loop sensors, video camera sensors, or radar/LiDAR sensors that can yield sparse or weather-dependent data. Finally, infrastructure may be re-designed in an attempt to improve arterial traffic intersection performance. The specific objectives of re-designing arterial intersections are to 1) reduce travel time, 2) reduce energy/fuel consumption, 3) reduce greenhouse gas emissions (CO2, CO, HC, NOx, PM), and 4) to improve pedestrian and vehicle occupant safety.

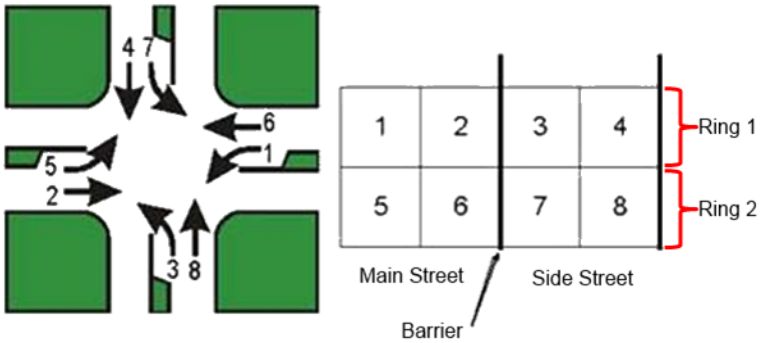

Figure 6: NEMA Dual-Ring Phasing Diagram [“Signal Timing on a Shoestring”, 2005]

The three general approaches mentioned are not mutually exclusive, and are in fact dependent on each other. Optimization of vehicle performance in a traffic network is ultimately constrained by the design of the traffic network. Similarly, the optimal operational efficiency and throughput of a traffic network is also ultimately constrained by the design of the traffic network. As a result, it is evident that vehicle performance and system operational performance should be optimized around an optimal traffic

system design. Instead of relying on conventional intersections such as the intersection shown in Figure 6, alternative intersection designs may be explored to form the basis of more efficient traffic systems.

In traffic engineering, the exploration of alternative traffic intersection designs is referred to as Unconventional Arterial Intersection Design (UAID). The goal of UAIDs is threefold: 1) to improve intersection efficiency by reducing the number of phases necessary for signal operation, 2) to improve intersection safety by reducing the total number of conflict points, and 3) to improve environmental sustainability of an intersection by reducing the number and severity of vehicle acceleration/deceleration events. The achievement of the goals results in 1) reducing the loss time due to transitioning between signal phases, 2) reducing the number and severity of collisions between vehicles in or near intersections, and 3) reducing vehicle emissions near population centers, respectively. Traditionally, the third criterion has been neglected to the extent that its inclusion constitutes what may be referred to as Eco-friendly Unconventional Arterial Intersection Design (Eco-UAID).

2.4.1.2 Description of Performance Metrics for Intersections

In order to assess the performance of various designs, a number of metrics may be employed based on the desired traffic objective. In the context of a single arterial traffic intersection, mobility may be quantified in terms of efficiency and capacity. The U.S. Department of Transportation (DOT) Federal Highway Administration (FHWA) has provided a table for quantifying intersection efficiency in terms of Level-Of-Service

(LOS), as shown in Table 1 below [Koonce, *et. al*., 2008]. An LOS of A indicates that the intersection is operating under close to free-flow conditions, whereas an LOS of F indicates that the capacity of the intersection has likely been exceeded. The assessment of the capacity of an arterial intersection is also dependent on subjective criteria, and may be calculated using various methods. The Highway Capacity Manual method involves theoretically estimating the capacity of a signalized arterial intersection based on the number of lanes, turning movements, and signal phase and timing operations [TRB, 2000]. In addition to estimating capacity, capacity may be approximately inferred based on vehicle trajectory data. Accordingly, actual capacity may be practically defined as the total traffic volume at which the average travel time is consistently 80 seconds greater than the free-flow travel time. In this case, the actual capacity varies based on the selected Origin-Demand (OD) matrix. The latter definition provides the benefit of being readily transferrable to assessing the capacity of UAIDs, and the benefit of providing an estimate based on simulated data rather than theoretical calculation. An ideal arterial intersection would not only have a high capacity, but also maintain a high LOS at traffic volumes nearing the capacity of the intersection.

Table 1: Motor vehicle Level-Of-Service (LOS) thresholds at signalized intersections

LOS	Control Delay per Vehicle (seconds per vehicle)
A	≤ 10
B	> 10-20
C	> 20-35
D	> 35-55
E	> 55-80
F	> 80

The safety of an arterial intersection may be roughly assessed by using conflict points. Conflict points include all points at which vehicle trajectories from different movements cross, merge, or diverge from each other. Conventional four-leg intersections have 16 crossing points, 8 merging points, and 8 diverging points, for a total of 32 conflict points, as indicated in Figure 7. If a vehicle "runs" a red light, crashes may occur at one or more of the intersection's conflict points. The reduction in the number of conflict points, especially crossing points, equates to an improvement in the safety of an intersection. In addition to vehicular occupant safety, the safety of pedestrians and bicyclists may also be considered in assessing the overall safety implications of a given intersection design. The pedestrian conflict points and bicyclist conflict points for a conventional four-leg intersection are shown in Figures 8 and 9, respectively.

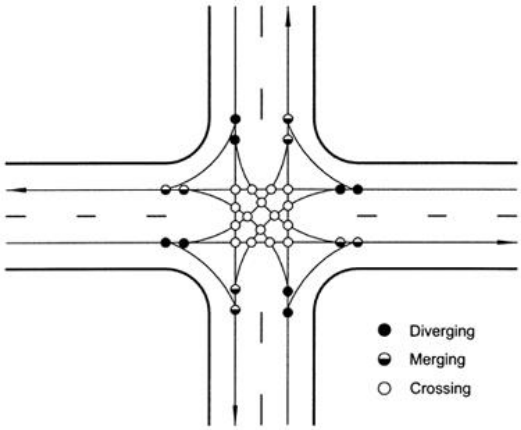

Figure 7: Conventional Intersection Vehicular Conflict Points [Rodegerdts *et al*., 2004]

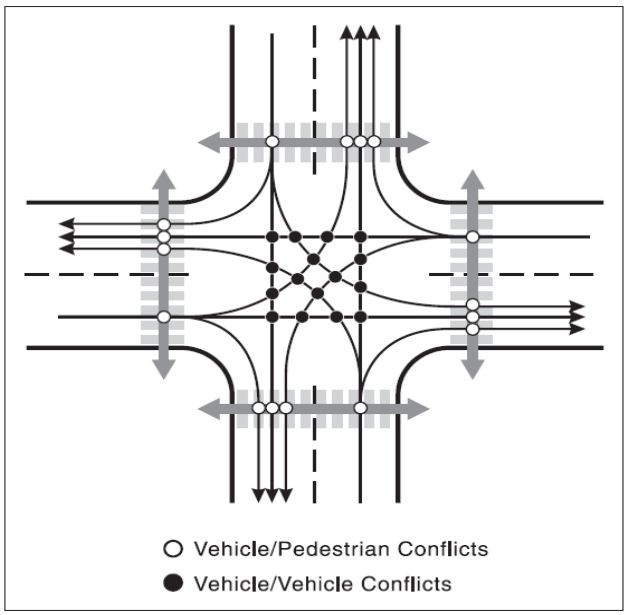

Figure 8: Conventional Intersection Pedestrian Conflict Points [Robinson, *et. al*., 2000]

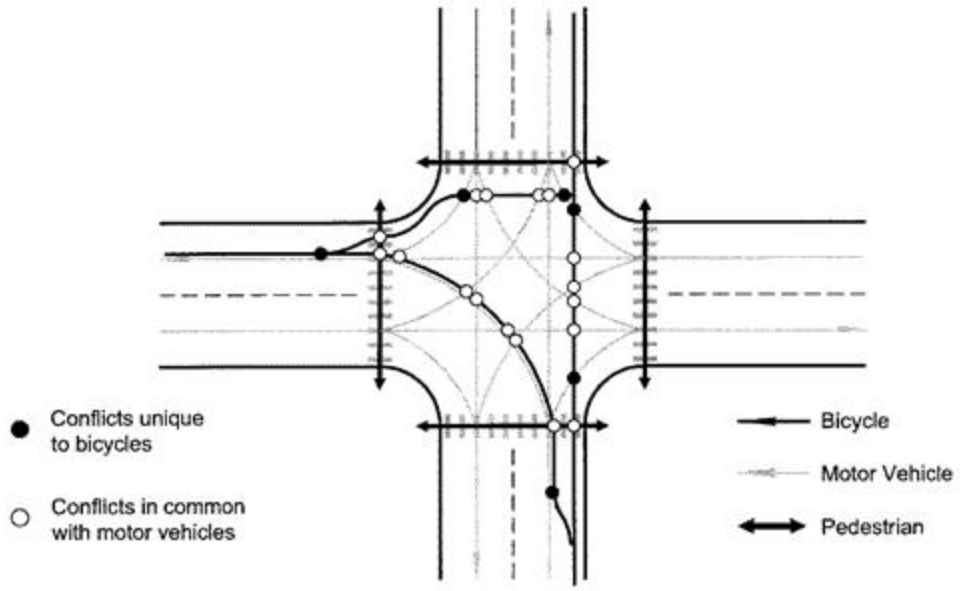

Figure 9: Conventional Intersection Bicyclist Conflict Points [Chandler, *et. al*., 2013]

The environmental performance of a given intersection design is often assessed in terms of vehicle emissions and fuel consumption. The primary vehicle emissions include carbon dioxide (CO_2), carbon monoxide (CO), hydrocarbons (HC), nitrous oxides (NO_x), and particulate matter (PM). In contrast to measuring fuel consumption, it is more challenging to attempt to directly measure vehicle tail-pipe emissions. As a result, emissions models coupled with microscopic simulation of vehicle trajectories provide a scalable and cost-effective means of estimating vehicle emissions for a given intersection design. In an urban setting, lowering vehicle emissions is of critical importance to preventing premature civilian fatalities due to exposure to pollutants.

2.4.2 Description of Existing Intersection Designs

Intersection designs may be classified based on whether the design is "at-grade" (i.e. without bridges), or grade-separated, and whether the design is signalized, (including traffic signals for operation), or unsignalized. In general, at-grade designs are preferable to grade-separated designs in terms of construction and maintenance costs. A variety of at-grade UAIDs have been proposed primarily over the past 30 years as alternatives to the conventional four phase 4-leg intersection. A majority of the designs are signalized and make use of secondary intersections to decrease the amount of traffic traveling through the primary intersection, as well as to decrease the total number of signal phases required. A few of the designs are unsignalized, having 0 signal phases, yet relying on a specified Right-Of-Way (ROW) to determine vehicle priority across multiple movements. The following sections will briefly describe 15 existing UAIDs, followed by a comparison of conflict points among the designs. Each section will illustrate and describe the design, briefly discuss the design's geometric and operational parameters, and specify known variations. The 15 existing UAIDs considered are as follows: 1) the double-T intersection, 2) the near-side jughandle intersection, 3) the far-side jughandle intersection, 4) the median U-turn intersection (MUT), 5) the restricted crossing median U-turn intersection (RCUT), 6) the super-street median crossover intersection (SSM), 7) the quadrant roadway intersection, 8) the split intersection, 9) the bowtie median intersection, 10) the parallel flow intersection (PFI), 11) the displaced left-turn intersection (DLT), 12) the double-crossover intersection (DXI), 13) the upstream signalized crossover (USC), 14) the round-about intersection, and 15) the hamburger intersection.

2.4.2.1 The Double-T Intersection

Many traffic intersections are designed to accommodate a minor street crossing a major arterial. The major arterial typically serves a greater volume of traffic than the minor street. As a result, geometric modifications are more often applied to minor approaches in order to improve mobility on the major street. For example, the double-T intersection dis-aligns the minor street legs to reduce the number of crossing points from 16 in a conventional intersection to 6 in the double-T intersection. Crossing points are typically categorized based on the movement types of the intersecting vehicle trajectories. Crossing points where both the intersecting trajectories are for through movements are referred to as "angle" crossing points, where the angle is typically at or near 90 degrees. In contrast, crossing points where at least one of the intersecting trajectories is from a left-turn are referred to as "left-turn" crossing points. In terms of safety, angle crossing points are considered more severe than left-turn crossing points due to the higher chance of a driver being blind-sided. In the case of the double-T intersection, not only are the number of crossing points reduced, but the type of the remaining 6 crossing points, left-turn, is of the less severe variety.

The double-T intersection essentially functions as two 3-leg intersections. It is important to note that a given intersection geometry may be operated in any one of several ways. In the case of the double-T intersection, the "t" intersections may be either signalized or use stop-signs on one or more approaches, depending on the amount of minor street volume. For a signalized double-T intersection, each of the "t" intersections

may operate using 3 signal phases instead of 4, due to the removal of minor through traffic. Vehicles on the minor left-turn movement must make a left-turn at the first "t", followed by a right at the second "t".

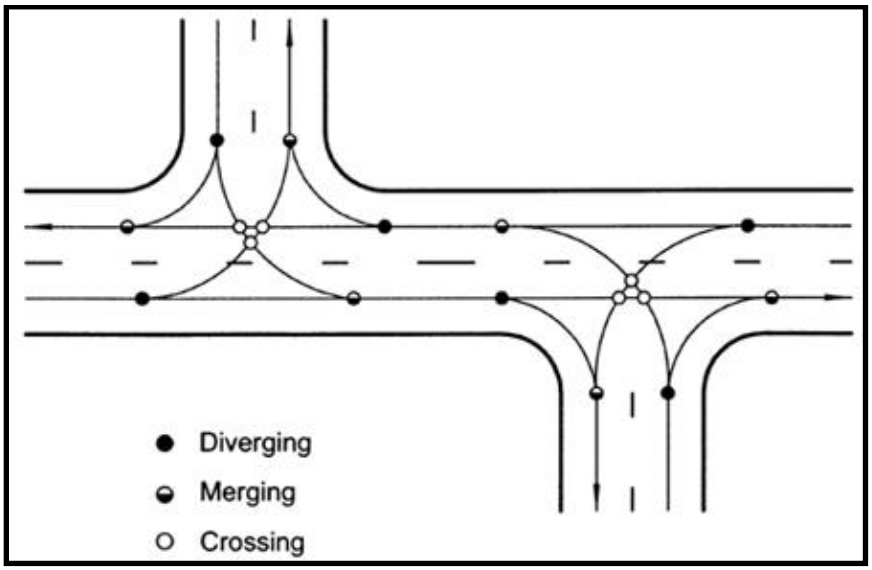

Figure 10: Double-T Intersection with Conflict Points Overlay [Rodegerdts *et al.*, 2004]

2.4.2.2 The Near-side Jughandle Intersection

Another method of removing left-turning vehicles from the main intersection is to employ "jughandles." Jughandles are one-way ramps that convert major left-turns into minor through movements as follows. Vehicles on the major street desiring to turn left must first turn right onto the jughandle, and then left on to the minor street before passing through the main intersection. The removal of a pair of left-turns from the main intersection reduces the overall number of conflict points, and permits the main

intersection to operate using 3 signal phases. The use of traffic signals on the two minor junctions is optional, and would require 2 signal phases.

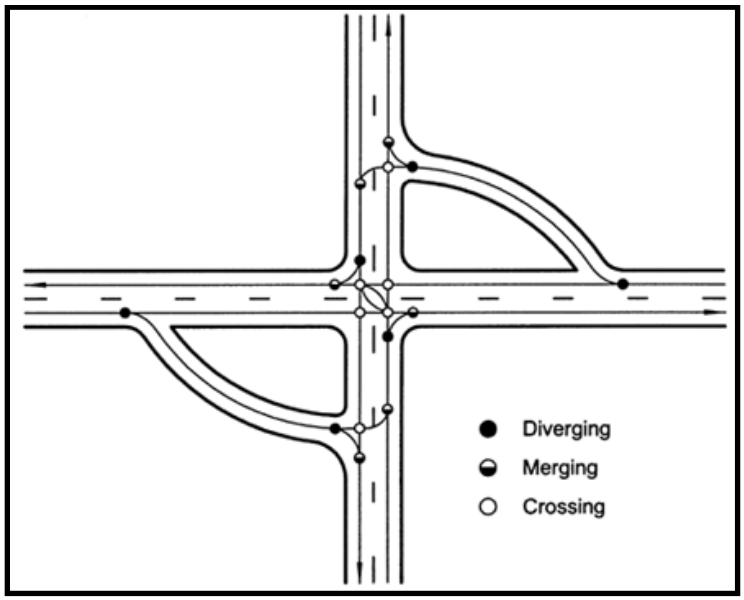

Figure 11: Near-side Jughandle Intersection with Conflict Points Overlay [Rodegerdts *et al*., 2004]

2.4.2.3 The Far-side Jughandle Intersection

The far-side jughandle is similar to the near-side jughandle with the exception that the major left-turn traffic merges onto the minor street after passing through the main intersection and looping around instead of turning onto the minor street prior to passing through the main intersection. The major left-turn traffic passes twice through the main intersection. Traffic turning right from the minor street is accommodated using a dedicated ramp. As a result, the minor right-turn traffic is removed from the main intersection. In contrast, a near-side jughandle removes the major right-turn traffic from the main intersection. As in the case of the near-side jughandle, the far-side jughandle

also operates using 3 signal phases. One advantage of the far-side jughandle over the near-side jughandle is that the far-side jughandle does not require additional minor intersections. One of the drawbacks of the far-side jughandle compared to the near-side jughandle is that the overall capacity of the main intersection is lower due to vehicles passing through the intersection twice. Whether a near-side or far-side jughandle is employed, vehicles turning left from the main street must travel a greater distance and potentially wait longer than left-turning vehicles at a conventional intersection. However, the overall system travel time may be reduced if the improvement in travel time for through movement vehicles outweighs the increase in travel time for the major left-turn movement vehicles.

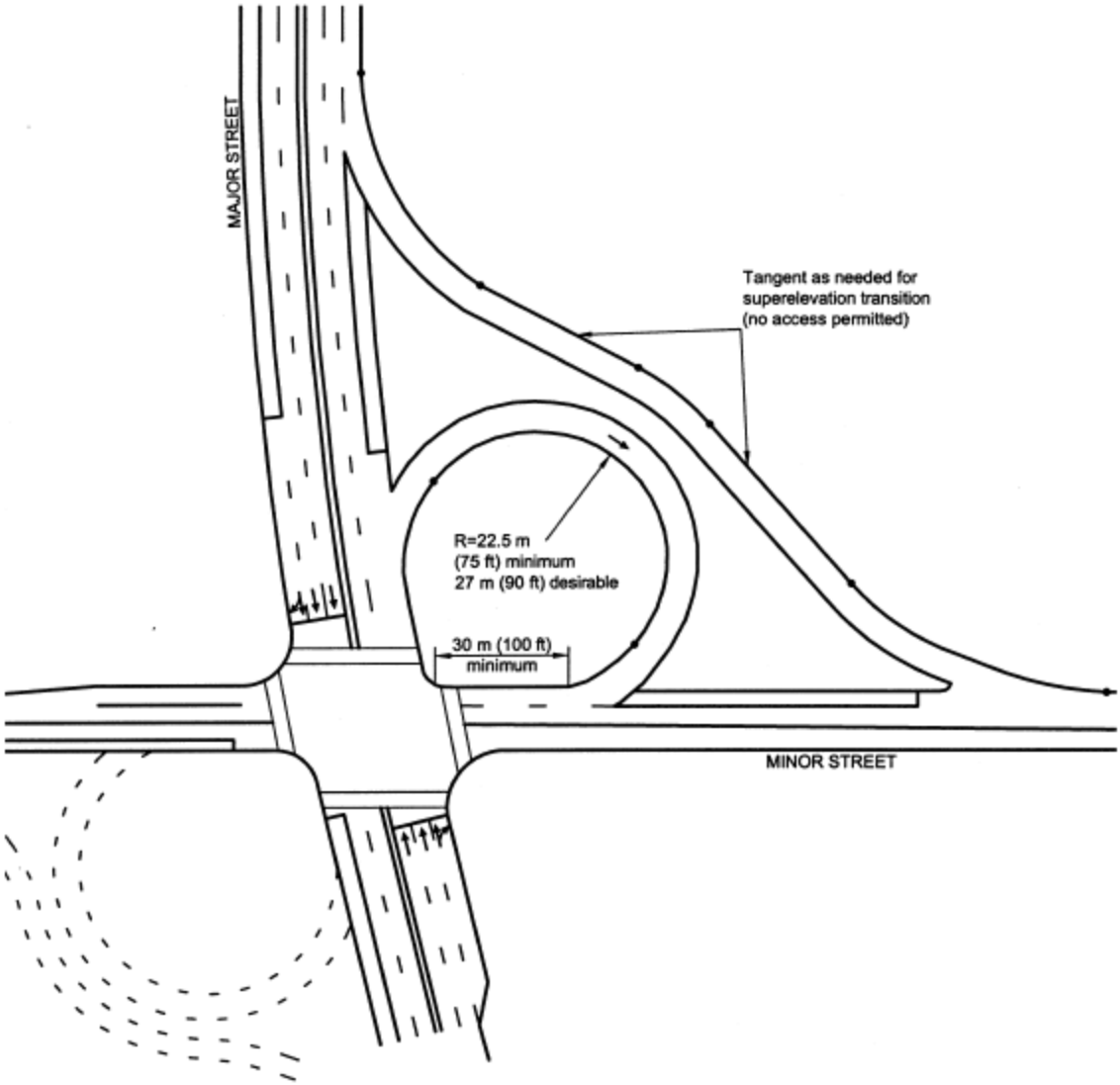

Figure 12: Far-side Jughandle Intersection [Rodegerdts *et al*., 2004]

2.4.2.4 The Median U-Turn Intersection (MUT)

One design which removes all of the turn traffic from the main intersection is the median U-turn (MUT) intersection [Rodegerdts *et al.*, 2004]. The median is typically placed on the major arterial, although an additional median may also be placed on the minor street. In the case of the former, major left-turn traffic first passes through the

intersection before making a U-turn to merge with opposing major street traffic, followed by a right-turn at the main intersection. In contrast, minor left-turning traffic must first turn right at the main intersection, and make a U-turn to merge onto the major street before passing straight through the main intersection. Due to the removal of all left-turn movements from the main intersection, only 2 traffic signal phases are needed for operations. The minor junctions may operate using either 2-phase signal control or yield signs. As in the case of jughandles, the overall travel time benefits are dependent on the relative ratios of through and left-turning traffic. High ratios of left-turning traffic will cause the intersection to perform worse than a conventional intersection. The primary advantage of the MUT intersection is the improvement in safety due to cutting the total number of conflict points in half relative to a conventional intersection. One potential drawback of the MUT intersection is that vehicles may experience a greater number of stops in passing through the intersections. A higher stop-rate is correlated with additional fuel consumption and emissions [Kari *et al*., 2014].

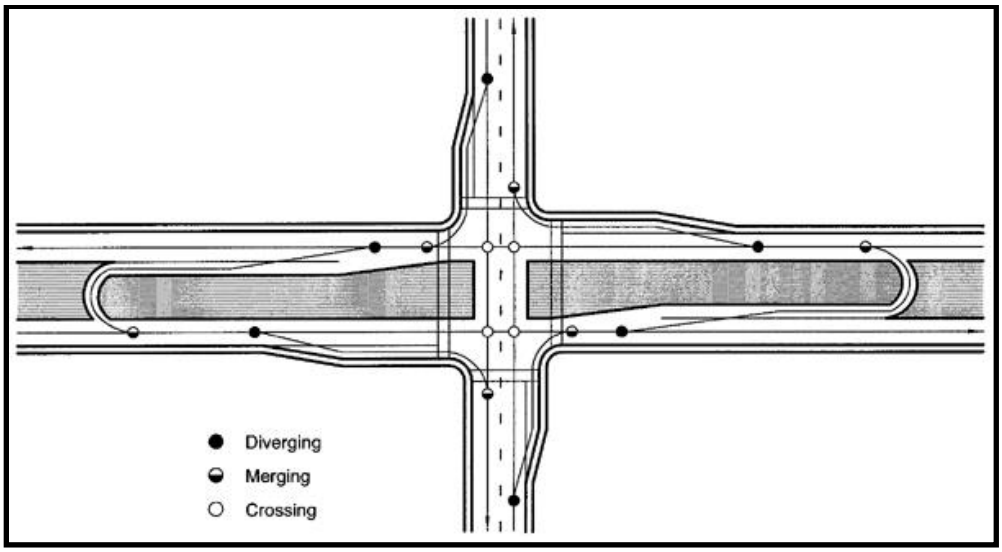

Figure 13: Median U-Turn Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004]

2.4.2.5 The Restricted Crossing Median U-Turn Intersection (RCUT)

The restricted crossing median U-turn (RCUT) intersection, sometimes referred to as the "unconventional MUT", [Esaway & Sayed, 2011], is a variation of the MUT intersection that prohibits minor through movements through the main junction. As a result of prohibiting major left-turns, minor left-turns, and minor through movements at the main junction, the entire intersection may operate without the use of traffic signals. Yielding may be employed both at the U-turns and at the main junction on both sides of the median. Alternatively, 2-phase signal control may also be used to operate the intersection. One of the major advantages of the RCUT intersection is that there are zero crossing points. The overall number of conflict points is reduced from 32 to 8. One of the drawbacks is that drivers turning left will need to change lanes multiple times.

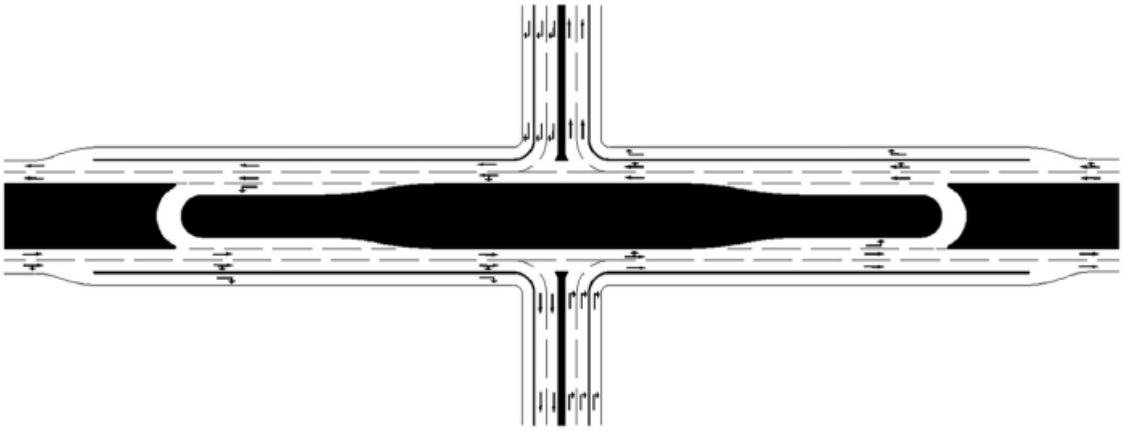

Figure 14: Restricted Crossing Median U-Turn Intersection with Conflict Point Overlay [Esaway & Sayed, 2013]

2.4.2.6 The Super-street Median Crossover Intersection (SSM)

The super-street median crossover (SSM) intersection prohibits minor through and left-turn movements at the main junction; however, major through and left-turn movements are permitted. The main junction of the SSM may operate using 2 signal phases. Similarly, the minor junctions may operate either using yield signs, or with 2 signal phases. If all three junctions are left unsignalized, the intersection is referred to as a J-turn intersection [Hummer, 1998]. The SSM also reduces the total number of conflict points in half relative to a conventional intersection. The average system travel time is lower for high proportions of major to minor traffic. Conversely, high ratios of minor traffic will increase the average travel time.

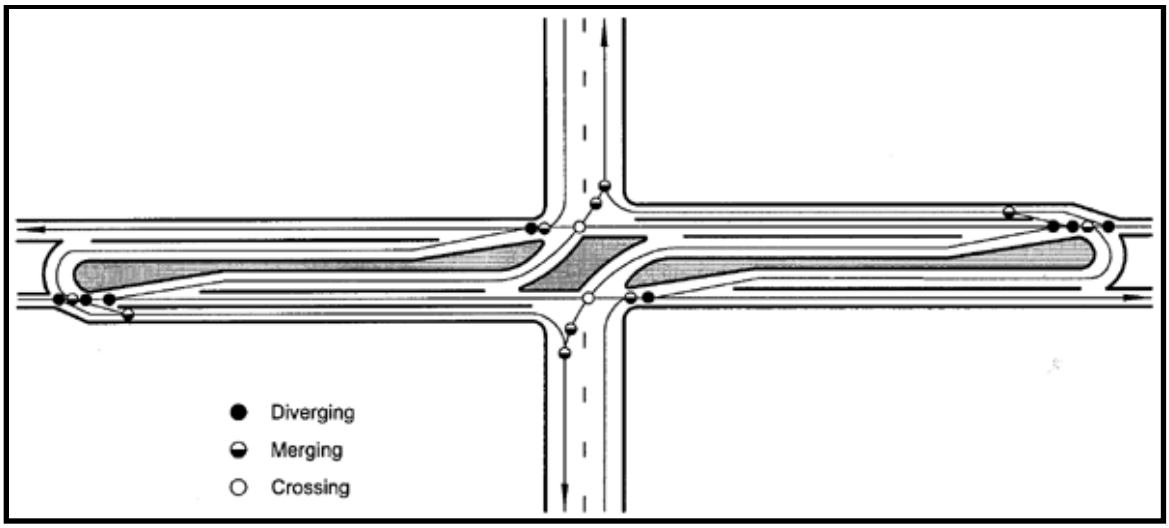

Figure 15: Super-street Median Crossover Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004]

2.4.2.7 The Quadrant Roadway Intersection (QR)

The quadrant roadway intersection removes all left-turn movements from the primary intersection by using a two-way connection in one of the four quadrants of the intersection. If the quadrant is located in the southwest corner of the intersection, as shown in Figure 16 below, then the major left-turn vehicles approaching from the west first turn right onto the quadrant, before turning left onto the minor street and finally passing through the main junction. In contrast, major left-turn vehicles originating from the east must first pass through the main junction, and turn left onto the quadrant before turning right onto the minor street. Minor left-turn vehicles originating from the north first turn right at the main junction, then left onto the quadrant and left out of the quadrant before once again turning right at the main junction. Minor left-turning vehicles originating from the south bypass the main junction by turning left entering the quadrant,

and left exiting the quadrant. The main junction operates using 2 signal phases due to the removal of all left-turns from the main junction. The minor junctions operate using 3 signal phases similar to the way 3-leg intersections operate. In order to facilitate movement into and out of the quadrant, the 3 traffic signals use coordinated signal timing plans.

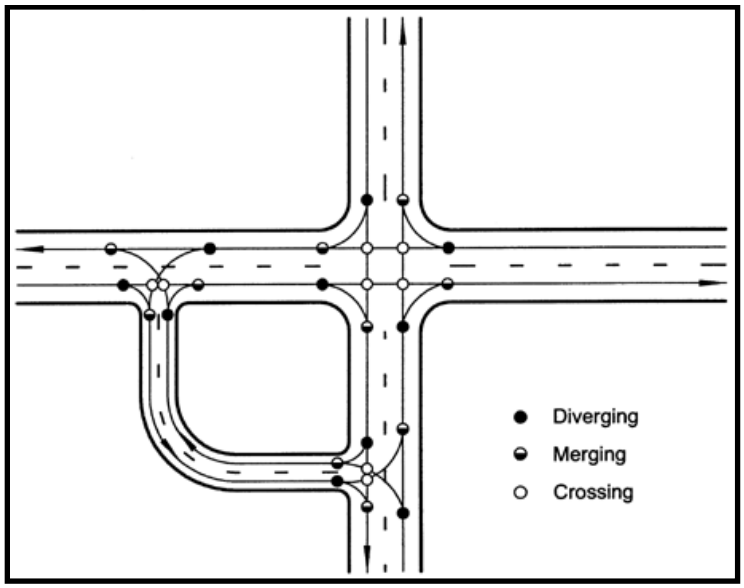

Figure 16: Quadrant Roadway Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004]

2.4.2.8 The Split Intersection

The split intersection splits the two minor approaches to the major street. The resulting intersection has 10 fewer conflict points than a conventional intersection and the conflict points are more spread out compared to a conventional intersection. The reduced number and reduced density of conflict points together reduce the chance of left-turn collisions. In addition, the two junctions can operate more efficiently than a conventional

intersection by using 3 signal phases instead of 4. The two junction's traffic signals are coordinated to allow vehicles to pass through both halves of the intersection with minimal delay.

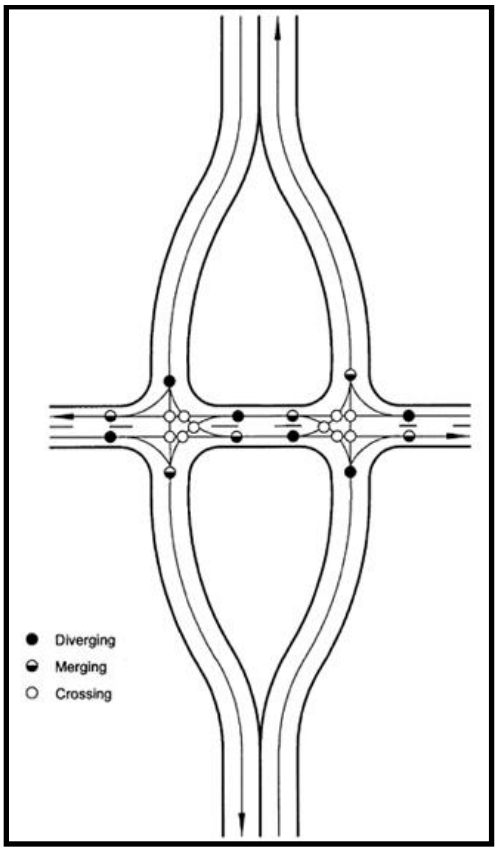

Figure 17: Split Intersection with Conflict Point Overlay [Rodegerdts *et al*., 2004]

2.4.2.9 The Bowtie Median Intersection

The bowtie median intersection restricts all left-turn movements at the main intersection by using two round-abouts placed on the minor street and functions similarly to a MUT intersection with the medians on the minor street. Left-turning vehicles from the main street first turn right onto the minor street, loop around the round-about, and

pass through the main intersection. In contrast, left-turning vehicles from the minor street first pass through the main intersection, and then loop around a round-about before turning right at the main intersection. The primary intersection may operate using 2 traffic signal phases and the round-abouts are unsignalized.

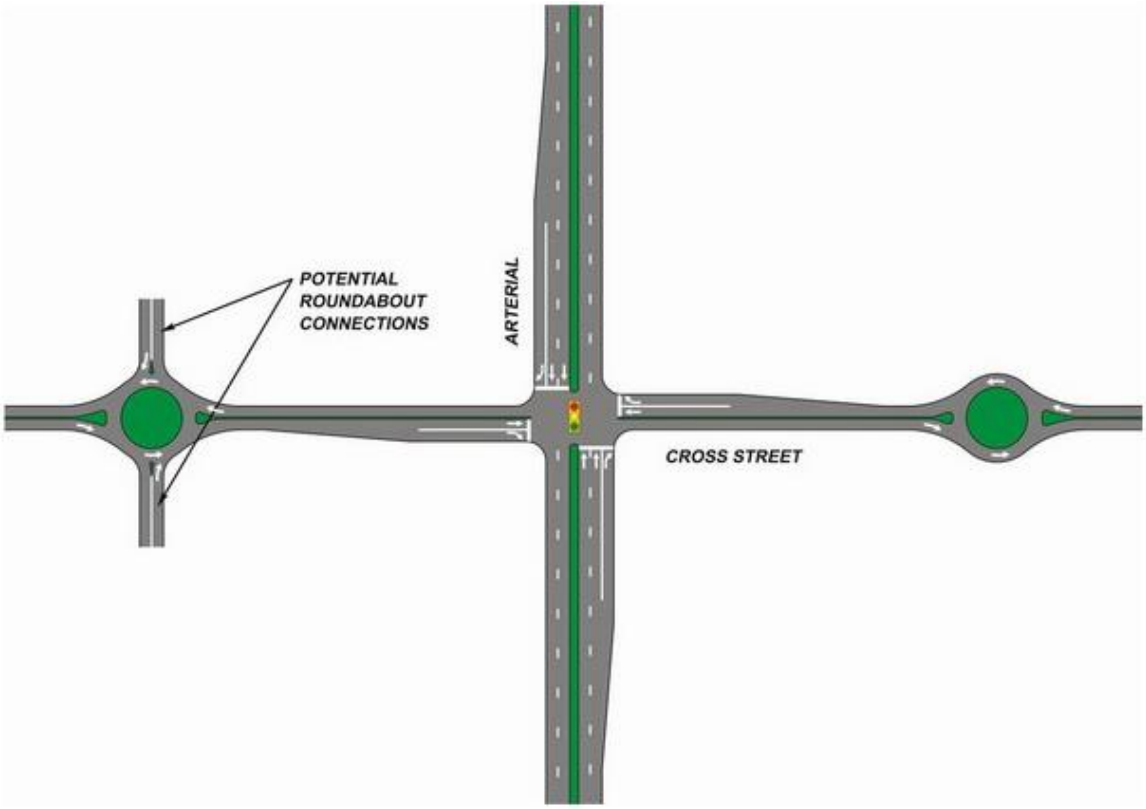

Figure 18: Bowtie Median Intersection with Conflict Point Overlay [MUID, 2015]

2.4.2.10 The Parallel Flow Intersection (PFI)

The parallel flow intersection (PFI) removes left-turning vehicles from the primary intersection by having left-turns crossover before the main intersection by using a bypass road parallel to through movement traffic [Parsons, 2009]. The resulting

intersection has 5 signalized junctions which all operate using 2 signal phases. The distance between the minor junctions and the main intersection dictate the optimal cycle length for coordinating progression through the overall intersection. Through movement vehicles pass through 3 signalized junctions, left-turning vehicles pass through 2 signalized junctions, and right-turning vehicles pass through a single signalized junction. The PFI was invented by Gregory Parsons in 2004 [Parsons, 2006].

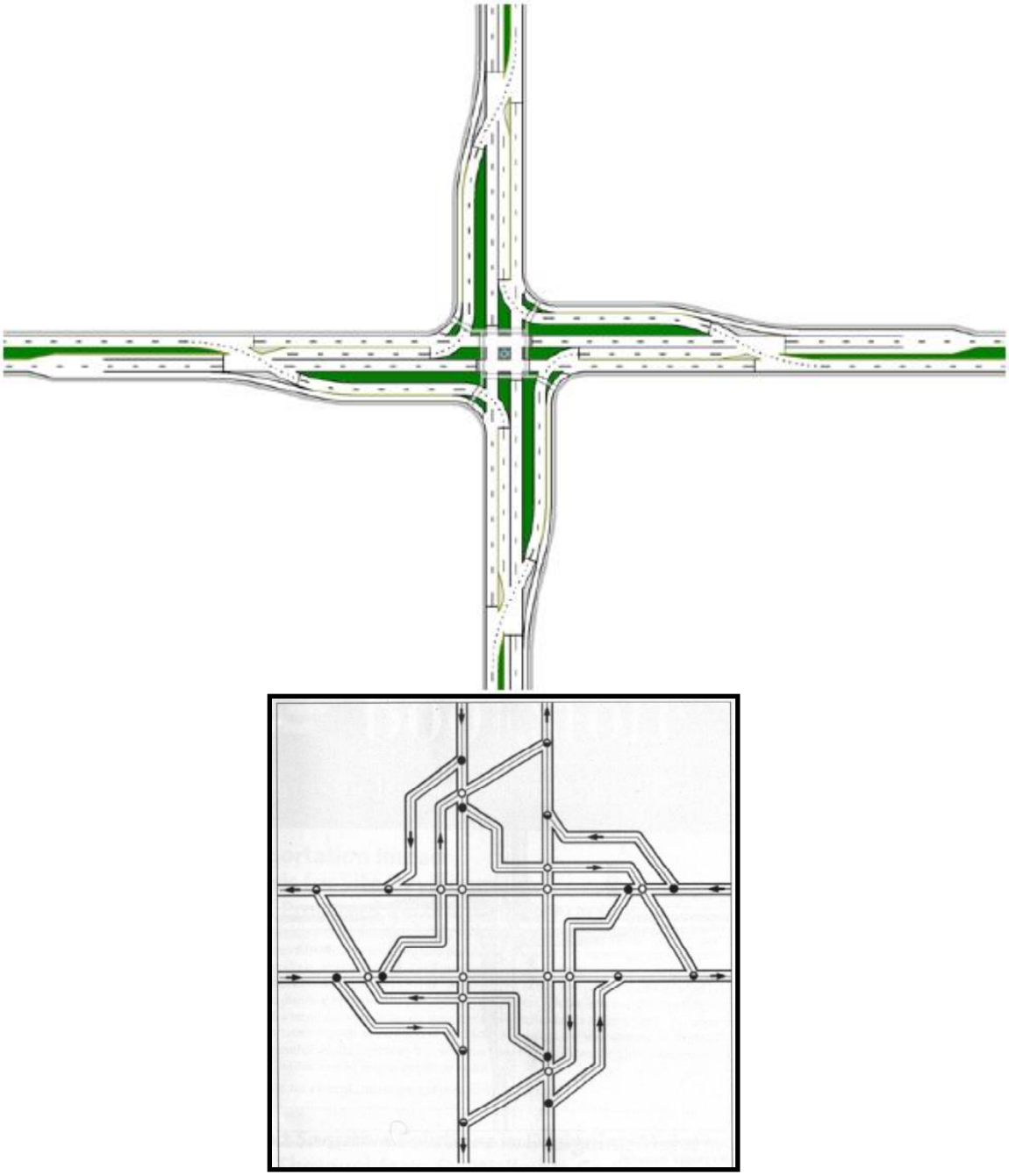

Figure 19: Parallel Flow Intersection (Top) with Conflict Point Overlay (Bottom) [Parsons]

2.4.2.11 The Displaced Left-Turn Intersection (DLT/CFI)

The Displaced Left-Turn (DLT) intersection is similar to the PFI with the exception that the left-turn movement crosses over the opposing through movement upstream of the main intersection at one of the minor intersections instead of crossing over the opposing through movement just upstream of the main intersection. Like the PFI, the DLT also has 5 junctions: 1 main intersection, and 4 minor intersections. Each of the junctions operates using 2 signal phases and the signal timing is coordinated to enable vehicles to pass through multiple junctions with minimal stopping. The DLT intersection was originally invented in the 1980s by Francisco Mier and Belisario Romo who patented a grade-separated version of the design in 1987 [Mier and Romo, 1991]. Mier and Romo called the design a Continuous Flow Intersection (CFI) due to the observation that there was always at least 1 traffic movement in motion, compared to a conventional intersection where all vehicles are stopped during the all-red signal phase. The DLT is also referred to as a crossover displaced left-turn (XDL) intersection [Esaway & Sayed, 2013].

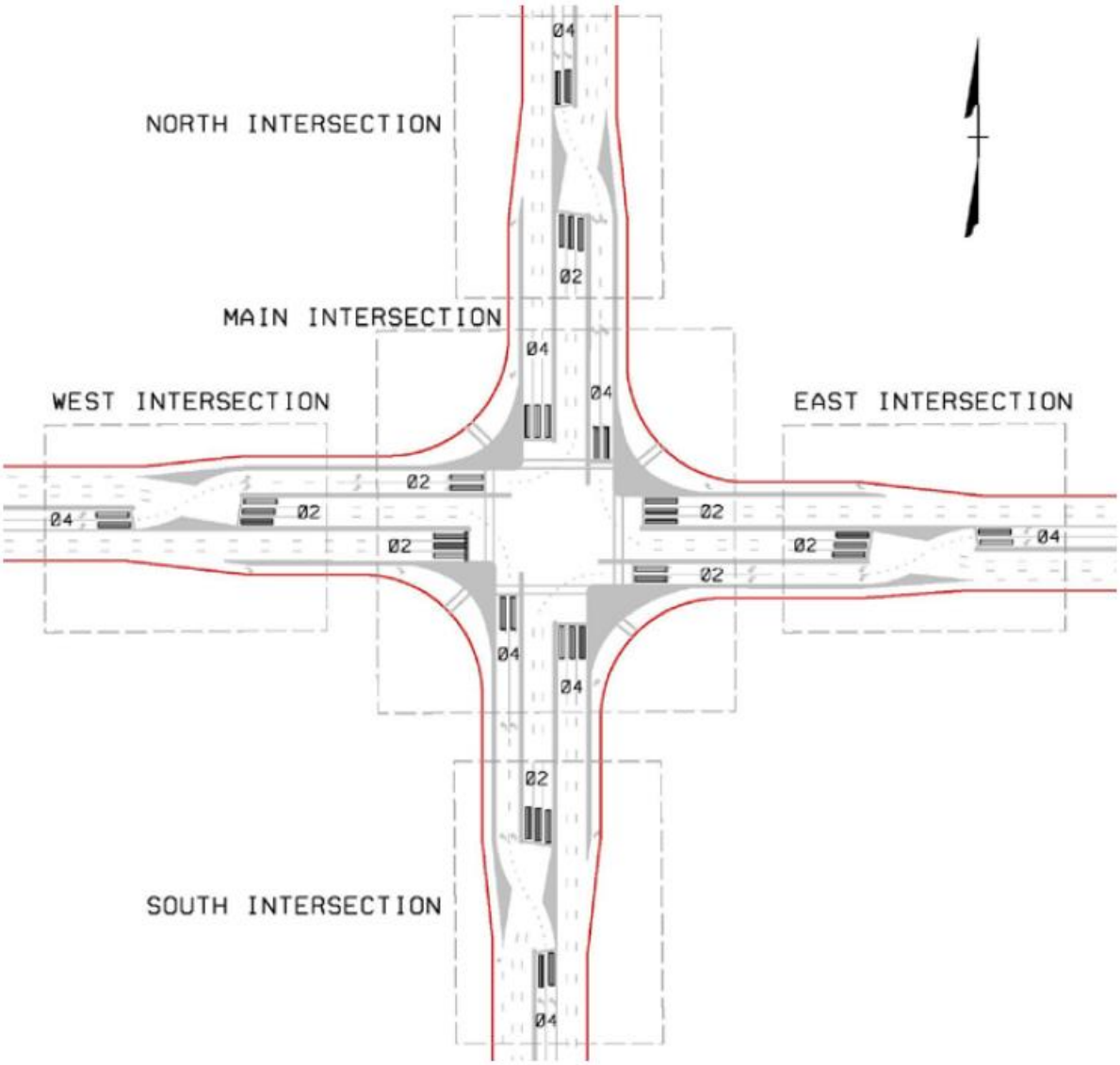

Figure 20: Displaced Left-Turn Intersection [Hughes *et al*., 2010]

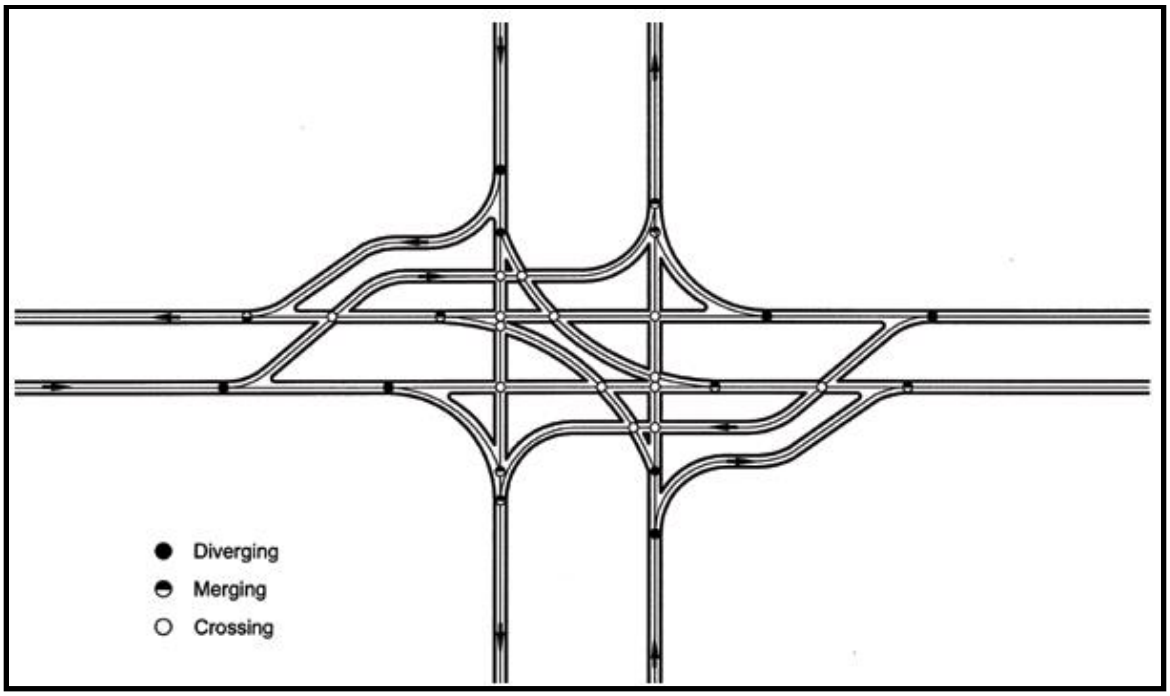

Figure 21: Displaced Left-Turn Intersection Conflict Point Overlay [Rodegerdts *et al*., 2004]

2.4.2.12 The Double-Crossover Intersection (DXI)

Unlike the PFI and DLT intersections, which cross over the left-turn movements, the Double-Crossover Intersection (DXI) crosses over both the left-turn and through movements for only the main arterial. The resulting intersection has 3 signalized junctions, with the main junction using 3 signal phases, and the 2 minor junctions using 2 signal phases. Through movement vehicles on the main road first crossover at the first minor junction, then proceed straight through the main junctions, and finally cross back at the second minor junction. The DXI was conceived by Gilbert Chlewicki who referred to the design as the Synchronized Split-Phasing (SSP) intersection [Chlewicki, 2003].

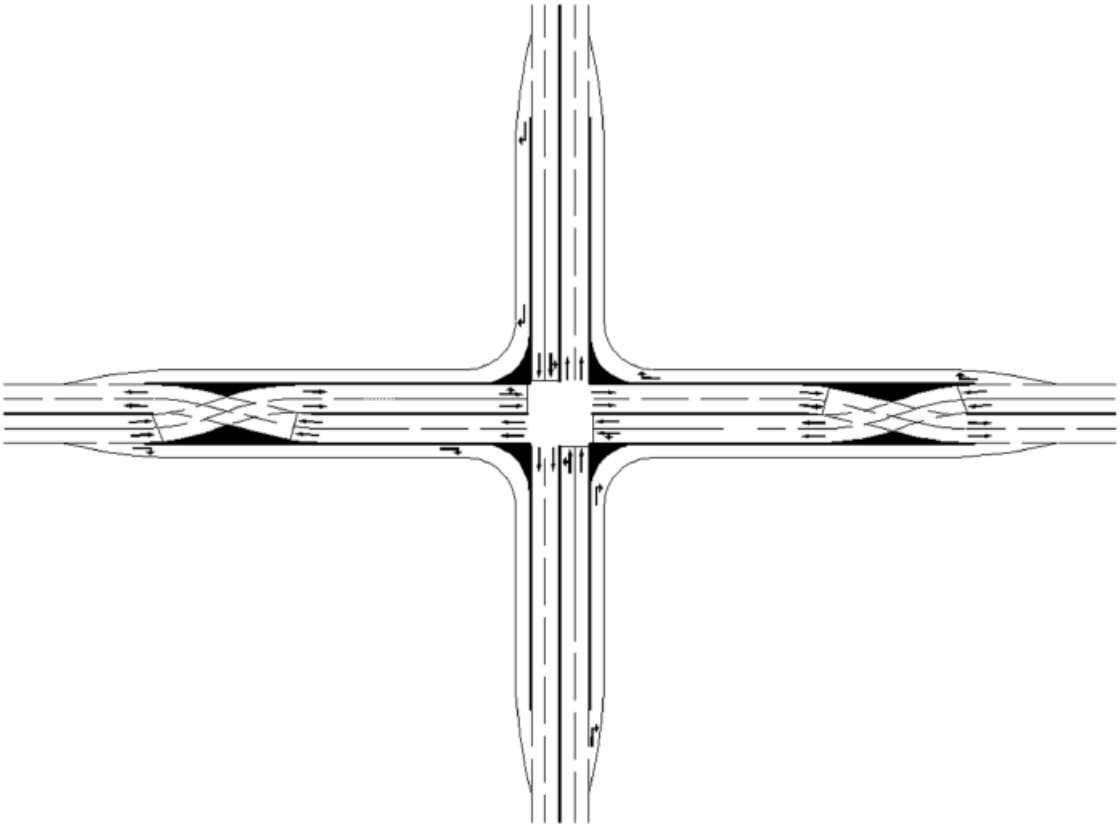

Figure 22: Double-Crossover Intersection with Conflict Point Overlay [Esaway & Sayed, 2013]

2.4.2.13 The Upstream Signalized Crossover Intersection (USC)

The Upstream Signalized Crossover (USC) intersection is a DXI with crossovers on both the major and minor streets. As with the DXI, the crossovers in the USC intersection are both upstream and downstream of the main signalized junction. All 5 junctions operate using 2 coordinated signal phases. The left-turn traffic turns left prior to reaching the main junction. The USC intersection is a logical extension of the DXI and was first published in 2005 [Tabernero *et al*., 2005]. The author independently conceived of the design in 2015 and entitled the intersection the Quadruple Crossover

Intersection (QXI). Although the name of Upstream Signalized Crossover adequately describes the upstream portion of the operations of the intersection, the name of Quadruple Crossover Intersection might be more appropriate.

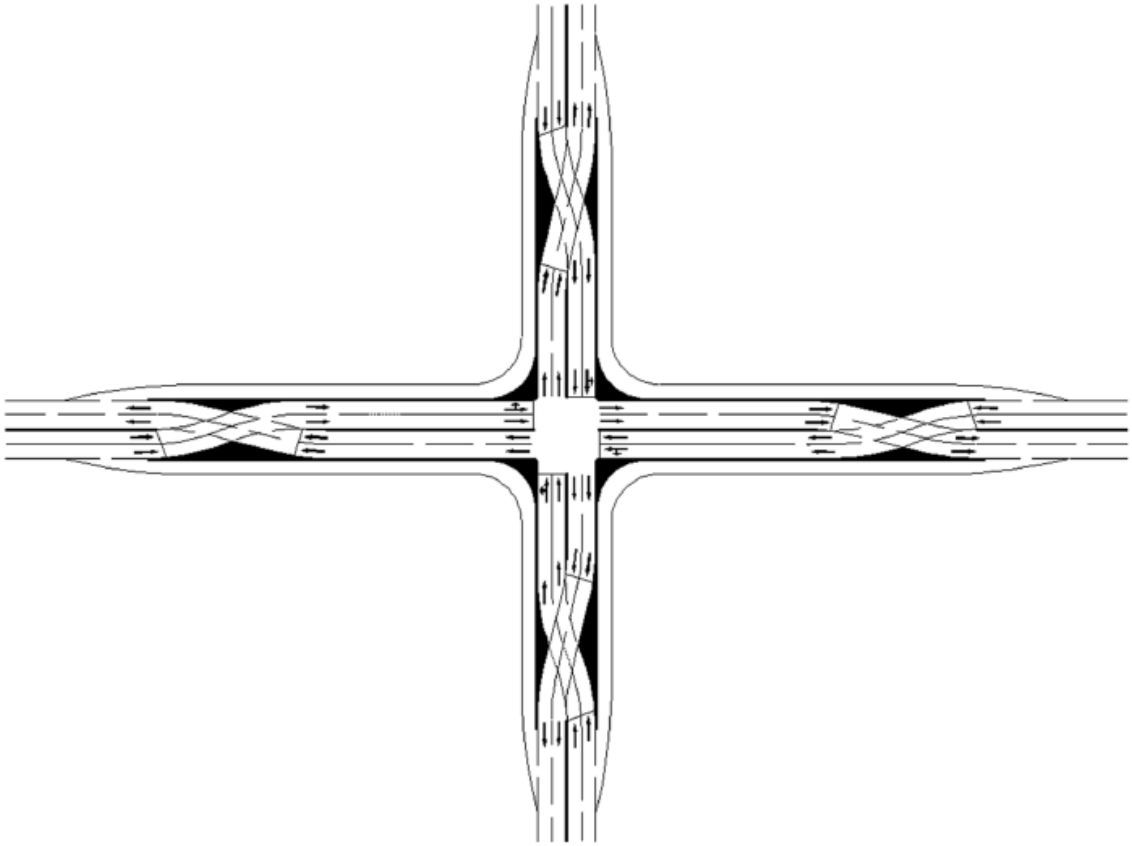

Figure 23: Upstream Signalized Crossover Intersection with Conflict Point Overlay [Esaway & Sayed, 2013]

2.4.2.14 The Round-about

Perhaps the most widely known UAID is the round-about. Round-abouts exist in numerous forms, but all have the common feature of prohibiting straight movement through the intersection. Generally, round-abouts have an island that is circular or at

least close to circular in shape. The basic round-about has 0 signal phases, operates on a yield protocol, and has a total of 8 conflict points. Importantly, round-abouts have 0 crossing points, thereby eliminating the chance of T-bone collisions. The only other known UAID with an identical number of conflict and crossing points is the RCUT intersection, which may be thought of as a round-about with the circular median extended along one axis.

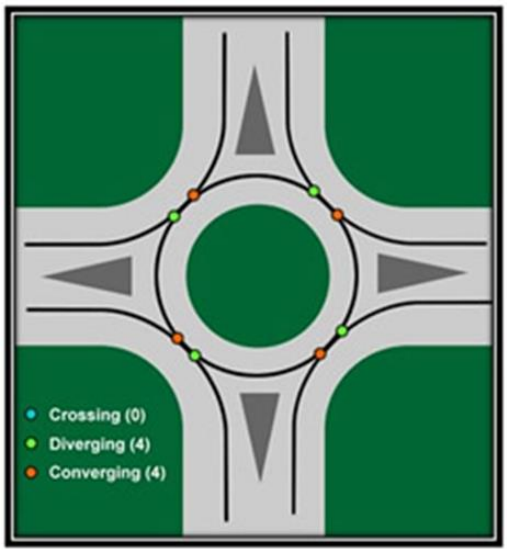

Figure 24: Round-about Intersection with Conflict Point Overlay [FHWA ("Safety Aspects of Roundabouts")]

2.4.2.15 The Hamburger Intersection

The hamburger intersection, or through-about, as it is sometimes referred to, is a variation of a round-about that permits through movement vehicles on the main arterial to cut through the circular median. Minor left-turning traffic also turns left through the circular median instead of going completely around the circular median. Unlike the

round-about, the hamburger intersection requires signalized control and operates using 2 signal phases on signals located on the main arterial road. According to recent simulation results, the hamburger intersection has a higher capacity than a round-about, but also a higher average delay than a round-about [Sangster *et al*., 2015].

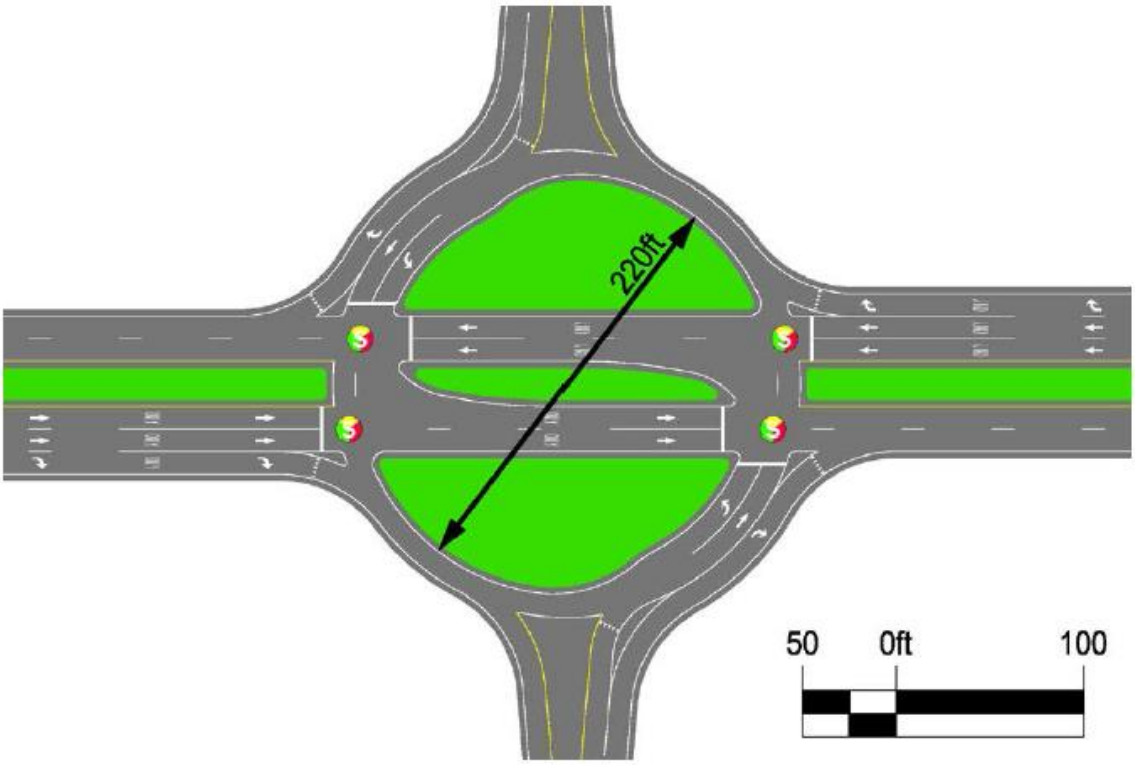

Figure 25: Hamburger Intersection with Conflict Point Overlay [Sangster *et al*., 2015]

2.4.3 Comparison of UAIDs

A simulation study simultaneously comparing the travel time performance and capacity of all 15 of the aforementioned UAIDs has not been conducted. Most, if not all of the existing studies analyze a small subset of similar UAIDs. The simulation results from different studies are not directly comparable due to the use of different

microsimulation software, different network areas, different number of lanes, different cycle lengths and offsets, different OD matrices, different evaluation periods, and different numbers of runs. One of the reasons why a single study has not yet rigorously simulated all 15 of the aforementioned UAIDs is due to the substantial amount of time necessary for coding, optimizing, and processing simulation data.

Nevertheless, the designs are comparable based on some of the general geometric and operational features, as shown in Table 2. Nearly all UAIDs offset the traffic load at an intersection by using multiple smaller intersections. In order to avoid confusion, the individual intersections that comprise the overall intersection will be referred to as "junctions," and the overall intersection will simply be referred to as "the intersection." Most UAIDs retain a main junction, located at the physical intersection of the main arterial and the cross-street. The additional junctions present in most UAIDs are auxiliary in function, and may be referred to as minor junctions. In terms of intersection traffic operations, all of the UAIDs considered reduce the number of signal phases required by at least 1 phase relative to a conventional 4-leg intersection. Comma-separated values in the third column of Table 2 are used to indicate the number of signal phases for multiple junctions. Similarly, forward slashes indicate that there are multiple control methods available for the given junction. Round-abouts, RCUT intersections, and SSM intersections may operate without using signal phases, but do require additional ROW procedures such as yield signs. It is also important to note, that several methods of operational control are often available for a given geometric design. For example, the RCUT intersection and the SSM intersection each may be controlled using yield signs or

2-phase signal control. Furthermore, every intersection design shown in the table below may be controlled using stop signs and yield signs instead of traffic signals. The selection of the method of operation is typically site specific, depending on the modes and volumes of traffic present. Generally, traffic signals are applied in situations with relatively high traffic volumes, as stop-sign controlled intersections have lower capacity than traffic-light controlled intersections.

Table 2: Geometric and Operational Comparison of UAIDs

Intersection Geometry	Number of Junctions	Number of Signal Phases	Additional Right-Of-Way
Conventional 4-leg	1	4	
Double-T	2	3, 3	
Near-side Jughandle	3	2, 3, 2	
Far-side Jughandle	1	3	
MUT	3	0/2, 2, 0/2	Yielding optional
RCUT	3	0/2, 0/2, 0/2	Yielding/Stop-signs optional
SSM	3	0/2, 0/2, 0/2	Yielding optional
Quadrant roadway	3	3, 2, 3	
Split intersection	2	3, 3	
Bowtie Median	3	0, 2, 0	Yielding
PFI	5	2, 2, 2, 2, 2	
DLT	5	2, 2, 2, 2, 2	
DXI	3	2, 3, 2	
USC	5	2, 2, 2, 2, 2	
Round-about	1	0	Yielding
Hamburger	2	2, 2	Yielding

As stated earlier, the safety of an intersection may be roughly assessed based on the number, type, and density of conflict points present in a given design. Table 3 catalogues the crossing, diverging, merging, and total conflict points for each of the UAIDs and the conventional 4-leg intersection. The next table, Table 4, shows a ordering of the intersections based on the number of conflict points. The only UAID with more conflict points than the conventional 4-leg intersection is the far-side jughandle. The intersections with the fewest conflict points are the RCUT intersection and the round-about.

Table 3: Conflict Point Comparison of Existing UAIDs

Intersection Geometry	Conflict Points	Crossing Points	Diverging Points	Merging Points
Conventional 4-leg	32	16	8	8
Double-T	18	6	6	6
Near-side Jughandle	26	10	8	8
Far-side Jughandle	36	24	6	6
MUT	16	4	6	6
RCUT	8	0	4	4
SSM	16	4	6	6
Quadrant roadway	30	10	10	10
Split	22	10	6	6
Bowtie Median	20	4	8	8
PFI	28	12	8	8
DLT	28	12	8	8
DXI	28	12	8	8
USC	24	8	8	8
Round-about	8	0	4	4
Hamburger	20	4	8	8

Table 4: Conflict Point Rank Ordering of UAIDs

Intersection Geometry	Conflict Points
Far-side Jughandle	36
Conventional 4-leg	32
Quadrant roadway	30
DLT	28
PFI	28
DXI	28
Near-side Jughandle	26
USC	24
Split	22
Hamburger	20
Bowtie Median	20
Double-T	18
MUT	16
SSM	16
RCUT	8
Round-about	8

The ranking of intersections based on the number of conflict points is a good starting point, but does not necessarily correspond to the relative safety levels of the intersections. In addition, using only the number of conflict points does not account for scenarios where vehicles may cross the same conflict point twice. A further refined ranking may be devised based on the number of conflict points vehicles cross when passing through a given intersection. The total number of routes through a given 4-leg intersection is 12. Each route may be indexed based on the origin and destination leg of the intersection. Representing the cardinal directions as N = North, E = East, S = South, and W = West, and considering up as North, the left-turns through a single intersection

may be specified as (N, E), (E, S), (S, W), and (W, N), where the notation (*origin leg*, *destination leg*) is used. Similarly, the through movements may be represented as (N, S), (E, W), (S, N), and (W, E). Lastly, the right-turn movements may be specified as (N, W), (E, N), (S, E), and (W, S). A table indicating the conflict points for each route for UAIDs is shown below (Table 5), succeeded by Table 6, which shows the revised ordering of the UAIDs based on total Origin-Destination (OD) route conflict points. Please note that the two following tables did not take U-turns into account.

Table 5: Origin-Demand (OD) Matrix Route Conflict Points for UAIDs

	Left-Turn Movement OD Pairs				Through Movement OD Pairs				Right-Turn Movement OD Pairs				
Intersection Geometry	(N,E)	(E,S)	(S,W)	(W,N)	(N,S)	(E,W)	(S,N)	(W,E)	(N,W)	(E,N	(S,E)	(W,S)	Total Route Conflict Pts.
Conventional 4-leg	8	8	8	8	8	8	8	8	2	2	2	2	72
Double-T	8	4	8	4	6	6	6	6	2	4	2	4	60
Near-side Jughandle	7	11	7	11	8	6	8	6	3	3	3	3	76
Far-side Jughandle	6	11	6	11	6	6	6	6	2	2	2	2	66
MUT	9	9	9	9	4	6	4	6	3	3	3	3	68
RCUT	6	6	6	6	4	4	4	4	2	2	2	2	48
SSM	8	4	8	4	6	6	6	6	2	4	2	4	60
Quadrant roadway*	12	10	8	10	8	6	8	6	4	2	4	4	82
Split	8	8	8	8	4	8	4	8	2	2	2	2	64
Bowtie Median	12	12	12	12	8	4	8	4	4	4	4	4	88
PFI	5	5	5	5	8	8	8	8	3	3	3	3	64
DLT	6	6	6	6	8	8	8	8	2	2	2	2	64
DXI	6	7	6	7	8	8	8	8	2	2	2	2	66
USC	6	6	6	6	8	8	8	8	2	2	2	2	64
Round-about	6	6	6	6	4	4	4	4	2	2	2	2	48
Hamburger	7	8	7	8	6	5	6	5	2	3	2	3	62

*Note that the quadrant roadway used for the analysis contained a single quadrant in the southwest corner of the intersection.

Table 6: Total OD Route Conflict Point Rank Ordering of UAIDs

Intersection Geometry	Conflict Points Traversed
Bowtie Median	88
Quadrant roadway	82
Near-side Jughandle	76
Conventional 4-leg	72
MUT	68
Far-side Jughandle	66
DXI	66
DLT	64
PFI	64
USC	64
Split	64
Hamburger	62
Double-T	60
SSM	60
RCUT	48
Round-about	48

In terms of ranking, most of the UAIDs were ranked similarly in Tables 3 and 5; however, the results for three intersections in particular highlight the utility of examining the conflict points traversed. Although the bowtie median intersection has 12 fewer conflict points than a conventional 4-leg intersection, vehicles traveling through the bowtie median intersection traverse 16 more conflict points than they would for a conventional 4-leg intersection. Similarly, although the MUT intersection was one of the most highly ranked intersections in terms of total conflict points, the intersection was below average for conflict points traversed. In contrast, although the far-side jughandle

intersection was the only intersection to have more conflict points than a conventional 4-leg intersection, the total conflict points traversed for the full OD matrix show a small improvement over the conventional 4-leg intersection. In addition to examining the total conflict points traversed, a similar analysis may be conducted for a subset of the conflict points, such as crossing points. Reducing the number of crossing points traversed while passing through an intersection is of particular interest for reducing the number of severe collisions at an intersection. The following two tables show the crossing points for each route for every UAID, and a ranking of the intersections based on crossing points traversed.

Table 7: Origin-Demand (OD) Matrix Route Crossing Points for UAIDs

	Left-Turn Movement OD Pairs				Through Movement OD Pairs				Right-Turn Movement OD Pairs				
Intersection Geometry	(N,E)	(E,S)	(S,W)	(W,N)	(N,S)	(E,W)	(S,N)	(W,E)	(N,W)	(E,N	(S,E)	(W,S)	Total Route Crossing Pts.
Conventional 4-leg	4	4	4	4	4	4	4	4	0	0	0	0	32
Double-T	4	2	4	2	2	2	2	2	0	0	0	0	20
Near-side Jughandle	2	5	2	5	5	3	5	3	0	0	0	0	30
Far-side Jughandle	4	9	4	9	5	5	5	5	0	0	0	0	46
MUT	2	2	2	2	2	2	2	2	0	0	0	0	16
RCUT	0	0	0	0	0	0	0	0	0	0	0	0	0
SSM	3	1	3	1	1	2	1	2	0	0	0	0	14
Quadrant roadway*	4	4	4	4	4	2	2	4	0	0	0	0	28
Split	4	4	4	4	2	4	2	4	0	0	0	0	28
Bowtie Median	2	2	2	2	2	2	2	2	0	0	0	0	16
PFI	3	3	3	3	4	4	4	4	0	0	0	0	28
DLT	2	2	2	2	4	4	4	4	0	0	0	0	24
DXI	3	3	3	3	4	4	4	4	0	0	0	0	28
USC	2	2	2	2	4	4	4	4	0	0	0	0	24
Round-about	0	0	0	0	0	0	0	0	0	0	0	0	0
Hamburger	2	2	2	2	2	2	2	2	0	0	0	0	16

*Note that the quadrant roadway used for the analysis contained a single quadrant in the southwest corner of the intersection.

Table 8: Total OD Route Conflict Point Rank Ordering of UAIDs

Intersection Geometry	Crossing Points Traversed
Far-side Jughandle	46
Conventional 4-leg	32
Near-side Jughandle	30
Quadrant roadway	28
Split	28
PFI	28
DXI	28
DLT	24
USC	24
Double-T	20
Bowtie Median	16
Hamburger	16
MUT	16
SSM	14
RCUT	0
Round-about	0

The ranking results based on crossing points traversed are similar to the ranking results based on total conflict points. Unsurprisingly, of the UAIDs considered, the far-side jughandle has the most total crossing points (24), and the most crossing points traversed for all OD pairs (46). All of the remaining UAIDs exhibited at least some benefit over a conventional 4-leg intersection in terms of reducing the total crossing points traversed for all OD pairs. Examining the results of the three ranking methods, (total conflict points, conflict points traversed for all OD pairs, and crossing points traversed for all OD pairs), reveals that the round-about and RCUT intersection were tied

for first in all three rankings. In addition, the SSM intersection was second in all three rankings.

Relatively little research has been conducted regarding the environmental performance of UAIDs. Part of the reason why environmental assessments of UAIDs are typically left out of reported results may be because many UAIDs negatively impact the environment relative to conventional intersections. As previously shown in table 2, a majority of alternative intersection designs add additional minor intersections. The addition of minor intersections increases the likelihood of vehicles stopping multiple times at the overall intersection. Increasing the number of stops made by vehicles negatively impacts fuel consumption and vehicle emissions. Unfortunately, existing UAIDs were by and large designed only in terms of mobility and safety, without considering energy and environmental impacts. In contrast, the designs proposed later in the dissertation intentionally incorporate energy and environmental considerations into the design cycle, perhaps meriting classification as Eco-UAIDs.

3. Software Design

Traffic simulation software is often used to evaluate the performance and potential impacts of novel methods of traffic control relative to existing traffic control scenarios. Traffic simulation is both a logical step towards field-testing as well as a companion strategy for fully gaging the impact of innovative traffic systems. Incorporating simulation into the design cycle is a relatively low-cost option that permits faster design iteration in a safe virtual environment. Even after a traffic system design has reached maturation and is field-tested, simulation permits large-scale testing that might otherwise be impossible or prohibitively expensive to conduct in the field.

In order to simulate vehicular traffic, an appropriate scope for modelling must be selected. Traffic modelling approaches may be generally classified as macroscopic, mesoscopic, or microscopic, depending on how time and space are modelled [Burghout *et al.*, 2004]. Real-world traffic operates in a continuous space and continuous time environment. In contrast, modelled traffic is simulated in a discrete time environment, with macroscopic traffic models using discrete space [Hüper *et al*., 2009], and microscopic traffic flow models utilizing continuous space environments, as shown in Table 9 below. Macroscopic modelling treats traffic as a compressible fluid and models the overall flow, density, and speed of traffic. In contrast, microscopic modelling models the kinematics of individual vehicles and their interactions with each other and the surrounding infrastructure. Essentially, the choice between macroscopic and microscopic modelling involves a tradeoff between faster execution times and level-of-detail. A

microscopic traffic flow approach was selected on the basis of achieving the highest fidelity and granularity of results. Furthermore, the vast majority of traffic agencies at the local, state, federal, and research levels also employ microscopic traffic simulation software for improving system efficiency as well as urban planning.

Table 9: Space-Time Categorization of Traffic Modelling

	Continuous Space	**Discrete Space**
Continuous Time	Real-world Traffic Systems	
Discrete Time	Microscopic Traffic Flow Models	Macroscopic Traffic Flow Models (i.e. Cellular Transmission Models)

Microscopic modelling of vehicular traffic includes a number of components such as: 1) physical infrastructure modelling, 2) vehicle kinematic modelling, and optionally 3) emissions modelling. In this research, a software program called PARAMICS was used for modelling physical infrastructure and vehicle kinematics [Quadstone, 2016]. A second software called MOVES was integrated with PARAMICS to provide emissions modelling of simulated vehicles. The overall simulation software system diagram is shown in Figure 26 below, followed by descriptions of each module.

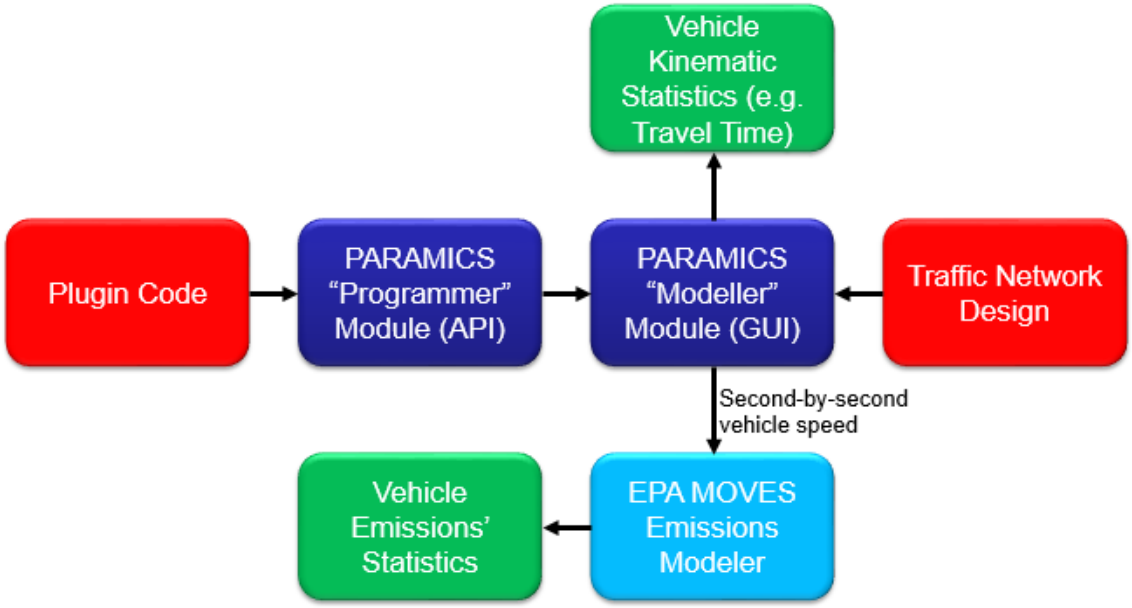

Figure 26: Microscopic Traffic Simulation System Diagram

The primary software program used for simulation was the Quadstone PARAMICS 6.9.3 software [Quadstone, 2016]. PARAMICS is a discrete-time, continuous-space microscopic traffic simulator which enables the simulation of models of custom or real-world traffic networks at the individual vehicle level. Traffic networks are modelled in PARAMICS software using "links" and "junctions." "Junctions" are nodes with unique IDs and 3D (X, Y, Z) coordinates. "Links" are road-way segments between two "junctions" and include attributes such as the number of lanes, the speed limit, and the lane width. In addition, "links" may be set to be one-way or bi-directional, and may be either arced or straight. PARAMICS provides a graphical user interface for constructing traffic networks with the "Modeller" software module.

Default vehicle behavior is determined on a step-by-step basis using default PARAMICS' car-following models, lane-changing models, road signs, and traffic signal

controllers. More specifically, at each time-step, the simulation calculates the next 3D position of a given vehicle based on the current vehicle 3D position, the current vehicle velocity, the current vehicle acceleration, the local roadway geometry, the relative positions and speeds of nearby vehicles, and roadway signs such as speed limit signs and traffic signals. These calculations are conducted once every time-step for every vehicle currently in the simulation and are continued until the specified duration of the simulation is reached.

Custom vehicle behavior is possible using the PARAMICS application programming interface (API) and the PARAMICS "Programmer" software module. The API provides a host of functions for programmatically interacting with the simulation using the Microsoft Visual C++ programming language. The PARAMICS "Programmer" software module enables plugin code to be first written in a "plugin.c" file in a Microsoft Visual C++ project. The code may then be compiled into a dynamic linked library (".dll") file which can be loaded into the PARAMICS "Modeller" module for specific traffic network projects. API functions are classified into one of four major categories: 1) Override functions, 2) Extending functions, 3) Set functions, and 4) Get functions. Override functions permit the user to adjust default PARAMICS behavior such as the built-in car-following and lane-changing models. Extending functions allow the user to add additional code to existing PARAMICS functions. For example, a user can choose whether to execute a certain segment of code at the beginning of a time step, or at the end of a time step. Set functions permit the user to change specific variable values for the simulation, such as vehicle speed or a link's speed limit. Get functions

allow the user to access variable values within the plugin code. Custom vehicle behavior is implemented by primarily using a sequence of "Get" functions to access the current state of the simulation, an application-specific algorithm to determine the desired next state for the simulation, and "Set" functions to programmatically alter variable values in the simulation.

The final component of the software system, the emissions modeler, was implemented using the Environmental Protection Agency's (EPA) MOtor Vehicle Emissions Simulator (MOVES) version 2010b tool [USEPA, 2011]. MOVES uses second-by-second vehicle speed and known vehicle characteristics such as mass and vehicle type to determine a vehicle's fuel consumption, and emissions of carbon dioxide (CO_2), carbon monoxide (CO), hydrocarbons (HC), nitrous oxides (NO_x), and particulate matter (PM). These environmental measures are important for assessing the carbon foot-print associated with current traffic systems and for developing sustainable strategies that focus on reducing fuel consumption and vehicle emissions.

4. Vehicle Enhancement Techniques

4.1 Eco-Cooperative Adaptive Cruise Control (Eco-CACC)

As described in section 2.2, prior research concerning CACC has focused on the operational and safety features of vehicle platooning. The following section addresses a number of research gaps including assessing the overall mobility and environmental impacts of CACC on freeway traffic. Specifically, the mobility and environmental impacts of permitting vehicles equipped with CACC to use a dedicated lane on a freeway are assessed, and the indirect effects on vehicles electing not to use CACC are also quantified. In addition, the CACC system presented is shown to have several parameters which are tuned to provide additional environmental benefits.

This section summarizes the application and results of modeling the Eco-CACC application for several freeway traffic networks using the PARAMICS (version 6.9.3) microscopic traffic simulation software and MOVES 2010b. A series of progressively complex networks were simulated, culminating in simulating Eco-CACC for the SR-91 E freeway in Southern California. The SR-91 E freeway was chosen because data on network geometry, traffic demand, and congestion levels were readily available, permitting calibration of the baseline scenario. Sensitivity analyses were performed on the following parameters to evaluate the environmental and mobility benefits that resulted from the introduction of this application:

- Traffic volume

- Penetration rate of Eco-CACC technology
- Triggering distance (defined below)
- Intra-platoon clearance.

A description of the Eco-CACC application is provided, followed by a detailed presentation of the selected Eco-CACC algorithm. Next, the selected freeway networks are presented, followed by simulation results. The remainder of the section is devoted to observations and discussions.

4.1.1 Application Description

The Eco-CACC application is an extension of ACC, (see section 2.2), that incorporates V2V as well as infrastructure-to-vehicle communication to provide mobility and environmental benefits to platoons of vehicles. Like ACC, Eco-CACC provides longitudinal control of a vehicle but still requires a driver to provide lateral control to the vehicle. Conventional ACC relies on sensors and actuators to detect and regulate the gap between the ego-vehicle (The term "ego-vehicle" will be used in the remainder of the text to denote a particular reference vehicle [the "self"-vehicle] being discussed in contrast to vehicles behind or in front of the current vehicle) and the vehicle directly ahead, whereas Eco-CACC adds communicated information from the vehicle directly ahead, and possibly the platoon leader, to permit shorter following gaps than are possible for ACC. Additional information from the infrastructure on network geometry and real-time traffic conditions further allows the speed of platoon leaders to be regulated to provide optimal environmental benefits for entire platoons.

4.1.2 Algorithm Design

To aid the subsequent Eco-CACC description, several terms are defined within the context of platooning, as Table 10 shows. The term "ego-vehicle" will be used in the remainder of the text to denote a particular reference vehicle (the "self"-vehicle) being discussed in contrast to vehicles behind or in front of the current vehicle.

Table 10: ITS Eco-CACC Definitions

	Front Bumper – Front Bumper	Back Bumper – Front Bumper
Time	Headway	Gap
Distance	Clearance	Spacing

The first half of the bumper pair refers to the bumper of the preceding vehicle and the second half of the bumper pair refers to the bumper of the current vehicle (behind the preceding vehicle).

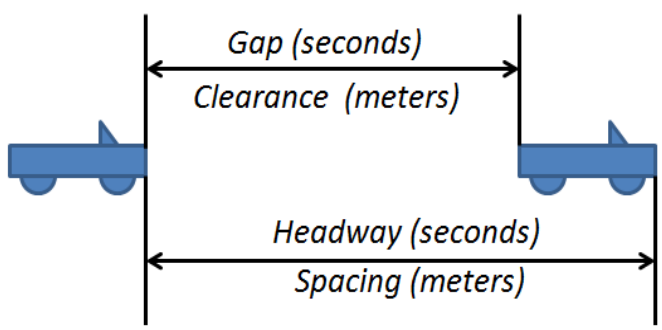

Figure 27: Illustration of Definitions

The core of the Eco-CACC algorithm is the definition of vehicle roles within a state machine, as Figure 27 shows. A state machine is a representation of a system in

which each possible behavior, or "state," is explicitly defined, along with the conditions that trigger transitions between states. The state machine may be viewed as a complete directed graph, in which any state is directly reachable from the current state. A vehicle is defined as leading a platoon (leader), following a platoon leader (follower), or not currently in a platoon ("none"). Unequipped vehicles are considered permanently part of the "none" state. OBE-equipped CVs in the "none" state may transition to being a follower if there is a vehicle ahead that is sufficiently close. Alternatively, an equipped vehicle in the "none" state may become a leader if a vehicle behind the ego-vehicle is within range to trigger platoon formation behavior. A follower may transition to the "none" state by either dropping off the tail of the platoon or by changing lanes. In addition, a follower vehicle may become a leader if the vehicle ahead was a leader and changed lanes. A leader may become a follower if the leader is overtaken by an equipped vehicle. Finally, a leader may transition to the "none" state if they transfer leadership to the vehicle directly behind the ego-vehicle.

The outer loop of conditions relates to maneuvers that occur within the platoon's lane and are referred to as longitudinal maneuvers. The inner loop of conditions relates to maneuvers that occur in preparation or during vehicle lane changing and are referred to as lateral maneuvers.

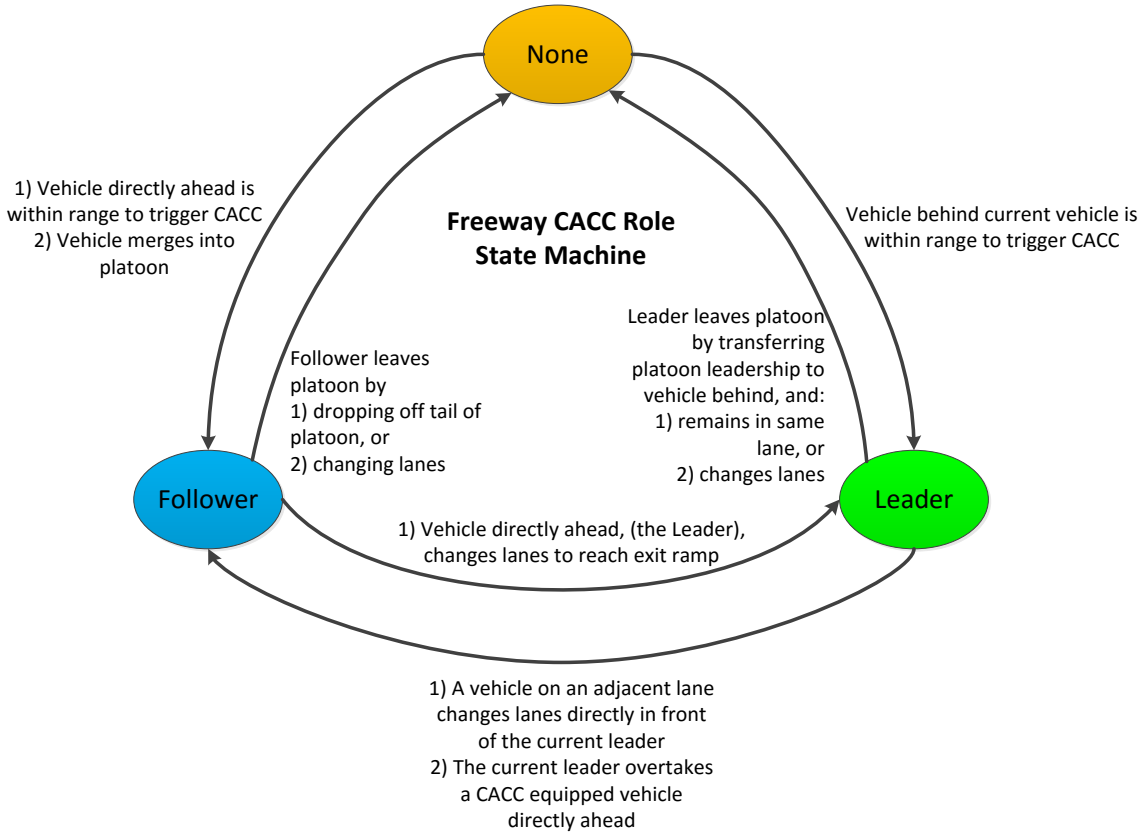

Figure 28: Freeway CACC Role State Machine

Figure 29 shows how the state diagram (in Figure 28) can be developed as an algorithm for the Eco-CACC application. The state flow diagram may be considered a three-stage process. In the first stage, if a vehicle is in a platoon and is approaching a freeway exit, a platoon-splitting protocol is executed. In the second stage, if an equipped vehicle is approaching a lane-drop area with an adjacent platoon in the target lane, a platoon-merging (single vehicle with platoon) protocol is executed. In the final stage, the vehicle state is updated based on proximity to surrounding equipped vehicles. Please note that the first two stages take priority over the third stage and result in changing the state of the ego-vehicle. After conducting a sensitivity analysis, the distance threshold

referenced in Figure 29 was set to 40 meters and is hereafter referred to as “triggering distance.”

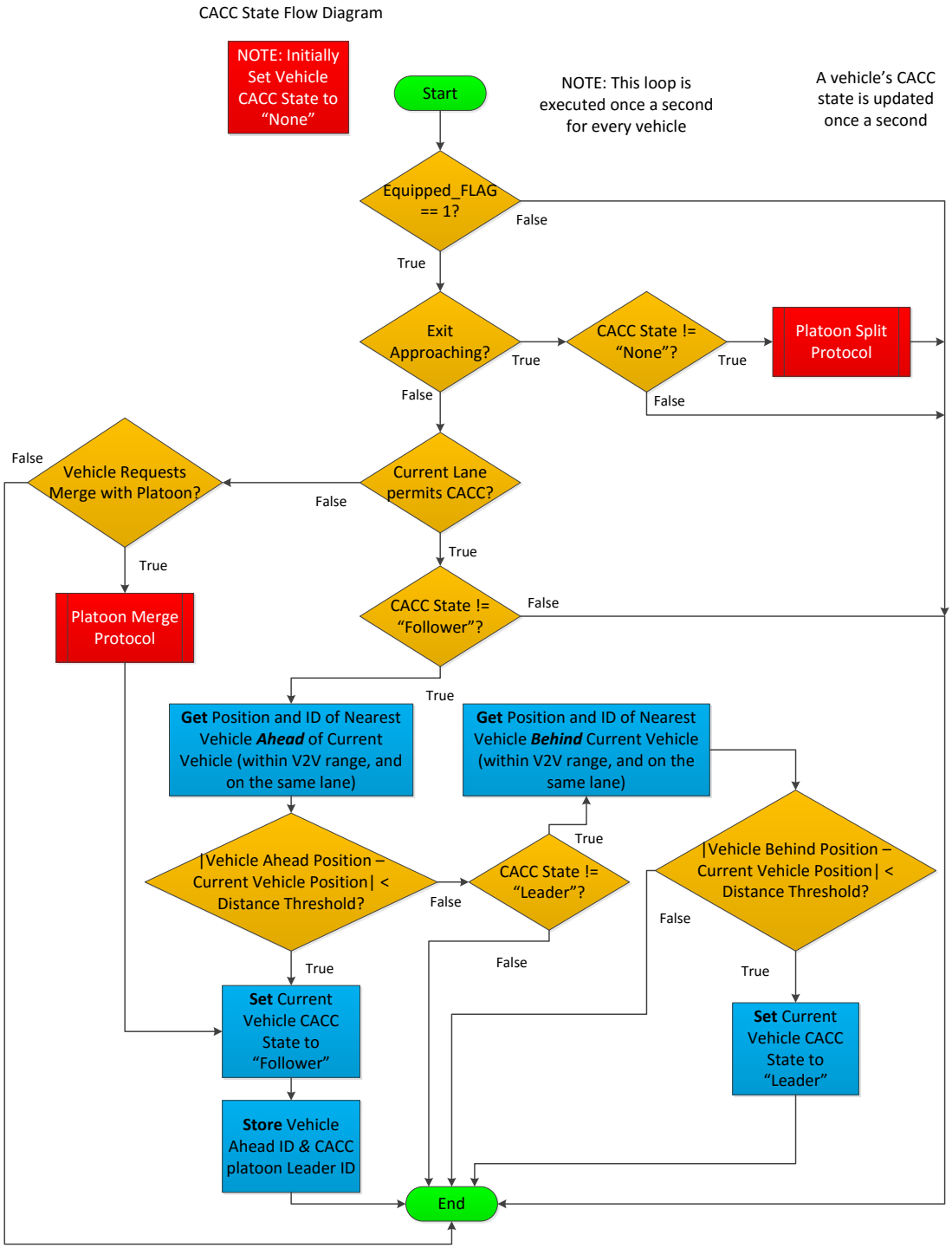

Figure 29: Freeway CACC State Flow Diagram

Vehicles within the leader or “none” states are permitted to be controlled by the car-following model in the simulation software. However, vehicles within the follower state have their own kinematics controlled by a custom gap controller. Numerous different gap controllers exist and are typically grouped in the general categories of spatial or temporal gap control. Spatial gap controllers seek to control the inter-vehicle distance between the ego-vehicle and the preceding vehicle. Temporal gap controllers seek to control the time between the ego-vehicle and the preceding vehicle. For the purposes of visualization, a spatial regulator was selected. As Figure 30 shows, the selected spatial regulator also incorporates the constraints of maximum acceleration and minimum deceleration. These acceleration constraints are not set by vehicle capabilities but for safety and driver comfort/acceptance during maneuvers. A later section in the chapter will explore how the acceleration constraints can be set to improve environmental performance. The presented spatial regulator in this section makes use of proportional control to regulate the error term (target clearance) using two parameters, R1 and R2. R1 is inversely related to the spatial gap closing rate, when the ego-vehicle is initially closing the gap. R2 is inversely related to the spatial gap opening rate and is used if the ego-vehicle overshoots the specified target clearance. The greater R1 or R2, the longer the vehicle will take to achieve the specified target clearance. For the simulation results presented in this section, R1 was selected to be 2, and R2 was selected to be 0.5. R2 is deliberately selected as lower than R1 for safety purposes. The remaining parameters were set as follows: Target clearance was set variously as 5 meters or 15 meters. Maximum acceleration and minimum deceleration were set to 3.5 meters and -7.5 meters

per second squared, respectively. Several additional controllers are shown in Appendix A for reference.

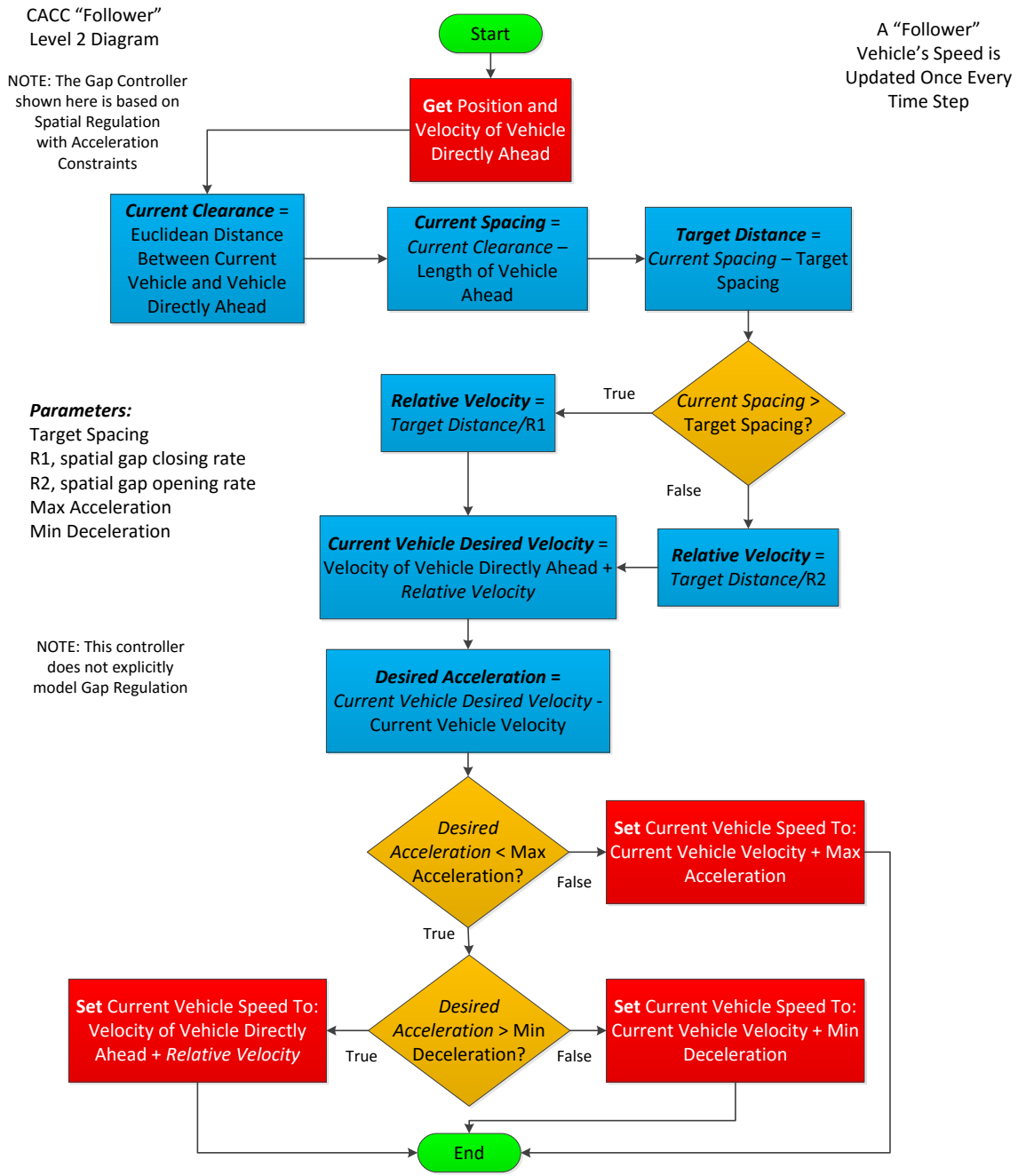

Figure 30: CACC "Follower" Level 2 Diagram, Spatial Regulator with Acceleration Constraints

4.1.3 Cooperative Maneuver Protocols

Depending on the initial set of assumptions, a variety of maneuvering protocols may be formed. The maneuvers introduced in this section are conducted at the individual vehicle level. Rather than having an entire platoon change lanes synchronously, the platoon is first split, and each vehicle changes lanes independently. No restrictions in terms of permitted vehicles are placed on the initial platoon formation (other than the vehicles being equipped for Eco-CACC). As a result, vehicles with different destinations (freeway exits) are permitted to participate in the same platoon. The selected heterogeneous route-based platoon formation stands in contrast with homogeneous route-based platoon formation, in which only vehicles with the same destination are permitted to participate in a given platoon. In homogenous route-based platoon formation, platoons may leave the freeway as an intact platoon. However, for heterogeneous route-based platoon formation, individual vehicles must leave their platoon in order to reach their desired exit. The developed splitting protocol shown in this section is based on heterogeneous route-based platoon formation.

As Figure 29 shows, the splitting protocol is executed when a vehicle in a platoon is close to its desired exit. Depending on where a vehicle is located in the platoon, one of three cases is performed. In each case, the splitting maneuver is a two-stage process. The first stage involves a relative longitudinal positioning of one or more vehicles. The second stage involves a lateral maneuver. The relative longitudinal positioning stage is accomplished by using the same controller used for following behavior (see Figure 30)

with a modulation of the target clearance parameter. The lateral maneuver of changing lanes is assumed to be executed by a human driver.

If a vehicle is the leader of the platoon, and leaves the platoon, then platoon leadership is passed on to the vehicle behind the current leader. Before starting the maneuver, the vehicle behind the current leader must acknowledge the leadership transfer request. Once the request is accepted, the vehicle behind the current leader increases its relative clearance with respect to the current leader by changing its target clearance parameter. As soon as the new target clearance is achieved by the vehicle behind the leader, the leader is free to change lanes and switches off its participation flag. By switching off its participation flag, the vehicle communicates to other vehicles that it does not desire to currently be a part of a platoon. In this case, the ego-vehicle prevents reformation of additional platoons (involving the ego-vehicle) on the way to its exit. The first column of Figure 31, (the left-most column), shows the case of a leader leaving its platoon.

The second column of Figure 31 shows the case of a vehicle at the tail of a platoon leaving its platoon. The tail of the platoon essentially drops off the back of the platoon before changing lanes.

The final two columns of Figure 31 show the case of a vehicle between the leader and the tail of a platoon leaving its platoon. The relative longitudinal positioning is applied to the ego-vehicle as well as the vehicle directly behind the ego-vehicle. The rationale behind having both vehicles participate in the first stage of the maneuver is to

create a safety gap both in front of and behind the ego-vehicle before control is transferred back to the human driver. The additional clearance not only contributes to enhanced safety but also helps the human driver to make the desired lane change.

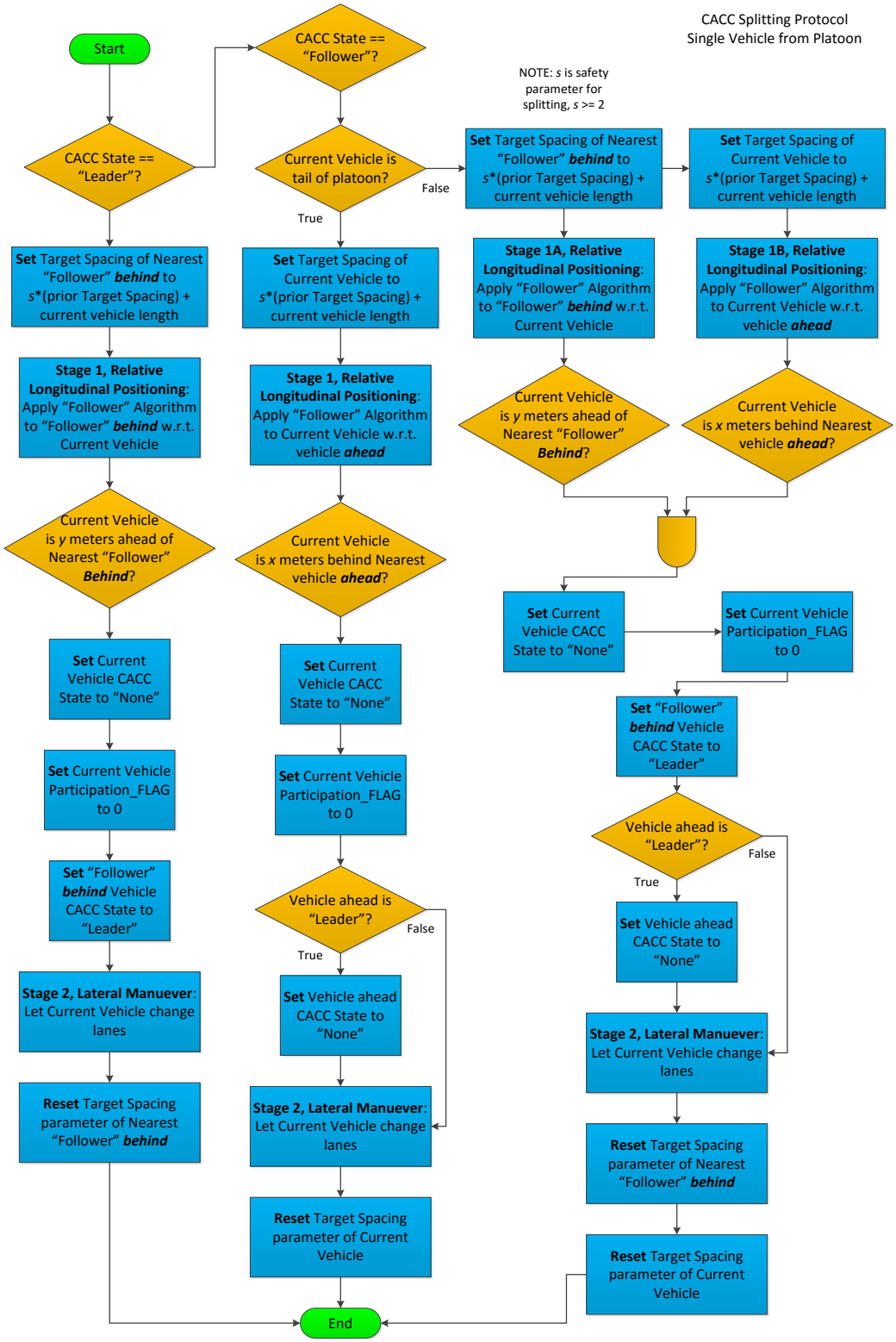

Figure 31: CACC Splitting Protocol, Level 2 Diagram

If an equipped vehicle is approaching a lane-drop area with an adjacent platoon in the target lane, the platoon-merging protocol that Figure 32 shows is executed. The maneuver is based on recognition of the two nearest vehicles in the adjacent lane. Once again, the maneuver is a two-stage process consisting of a relative longitudinal positioning stage and a lateral maneuver stage. The relative longitudinal stage involves two vehicles changing their relative positioning with respect to the nearest vehicle ahead of the ego-vehicle in the adjacent target lane. As seen in Figure 33a, vehicles 1 and 2 are part of a platoon and vehicle 3 is a vehicle attempting to merge with the platoon between vehicles 1 and 2. Vehicle 2 increases its target clearance with respect to vehicle 1 to allow sufficient room for vehicle 3 to merge with the platoon. In addition, vehicle 3 also drops sufficiently behind vehicle 1 in order to reduce the chance of colliding during the merging maneuver. Vehicle 3's knowledge of its longitudinal position relative to vehicle 1 is a combination of diagonal radar units, Global Positioning System, and CV technology. Once the relative positioning phase is complete (Figure 33b), the driver of vehicle 3 changes lanes and transfers longitudinal control to the Eco-CACC controller (Figure 33c). The term "Eco-CACC controller" refers to the physical device present on an equipped vehicle, which controls brake and throttle actions in response to the Eco-CACC algorithm previously introduced.

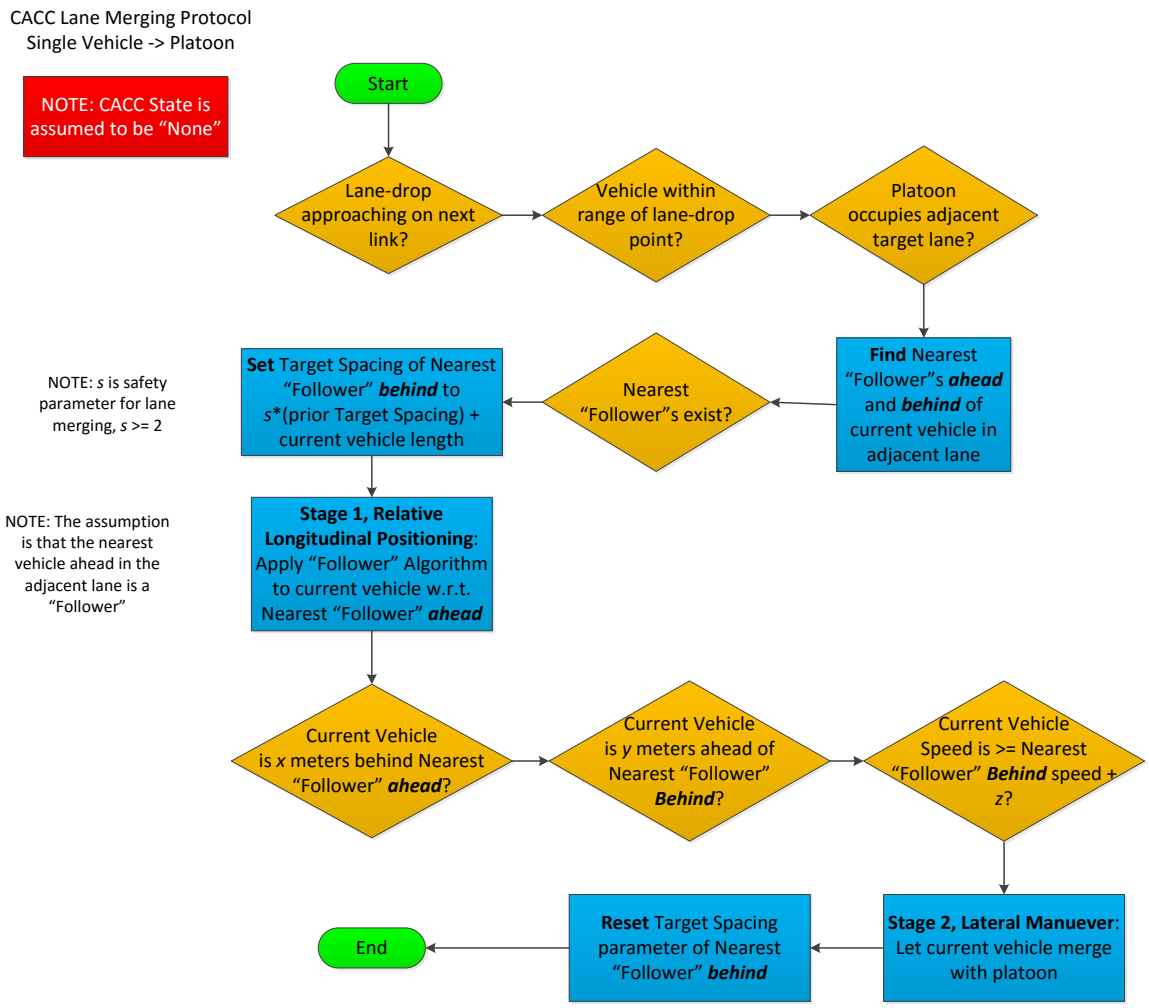

Figure 32: CACC Lane Merging Protocol, Level 2 Diagram

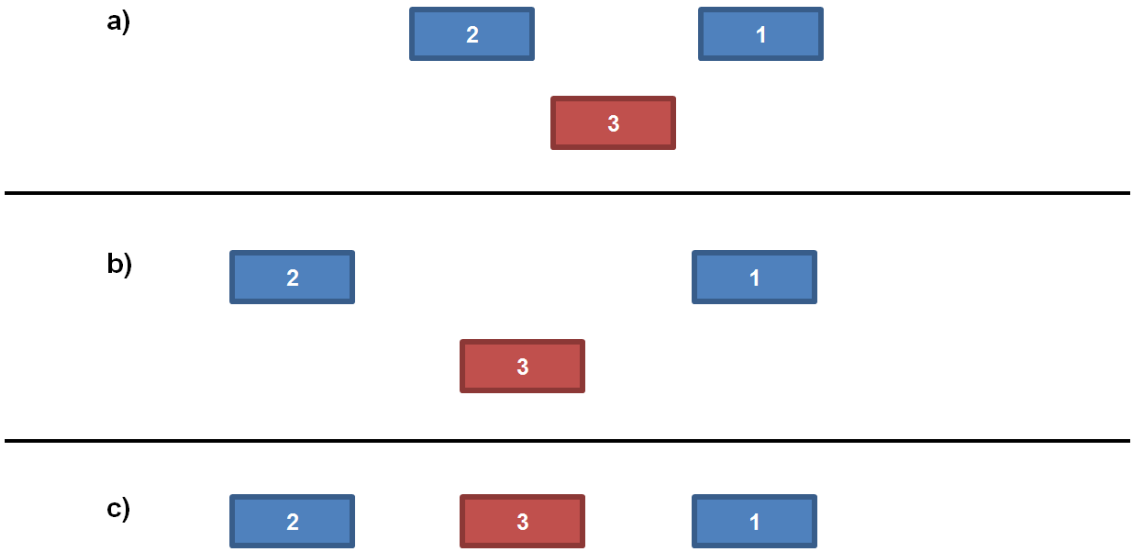

Figure 33: CACC Merging Protocol; a) Prior to Stage 1, b) Completion of Stage 1, c) Completion of Stage 2

4.1.4 Modeling Approach

Two main approaches exist for modeling CACC behavior within the microscopic traffic simulator PARAMICS. The first approach is to alter simulation parameters, such as headway and reaction time, to mimic the desired behavior of achieving lower gaps between vehicles. The second approach is to explicitly model CACC behavior using a state machine and vehicle controllers. The first approach has the advantage of being quickly and easily implemented within microscopic simulation. The second approach has the advantage of permitting direct control over vehicles within platoons and modeling more complex vehicle interactions. In terms of accurately modeling real-world behavior, explicitly modeling platoon behavior is superior to indirectly mimicking platoon

behavior. As a result, the second approach was selected and serves as the basis for the following modeling of the aforementioned Eco-CACC algorithm.

4.1.5 Eco-CACC Results

4.1.5.1 Simulation Networks Description

In order to simulate Eco-CACC in a microscopic simulation environment, the PARAMICS 6.9.3 software was selected along with EPA's MOVES 2010b software for modeling emissions. Detailed information about the software setup may be found in chapter 3. A series of three progressively complex networks were modeled in PARAMICS. The simpler networks were designed to test core elements of the developed Eco-CACC algorithm, whereas the most complex network was designed to test how Eco-CACC might work in a real-world freeway traffic network. The three networks are briefly described as follows.

4.1.5.1.1 Network #1: Hypothetical Freeway Segment with Lane-Drop

Network #1 consists of a 10 km straight stretch of a hypothetical freeway segment, with traffic traveling from left to right. The first 8.75 km consist of three lanes, followed by a 0.25 km section with two lanes, and a 1 km section with three lanes. None of the three sections are interrupted by on or off ramps, and all sections are given a road grade of zero. The last 1.5 km of the PARAMICS network are shown in Figure 34 below. The prior 8.5 km, not shown in the Figure, simply include a straight stretch of freeway consisting solely of three lanes.

Platoon formation was permitted on the two left-most lanes. In order to avoid platoons needing to merge with other platoons near the lane drop, platoon formation was restricted on the right-most lane. Individual vehicles on the right-most lane change lanes to the left when they approach the lane-drop area. If a platoon is in the middle lane and is blocking a vehicle needing to change lanes, the developed merging protocol is enacted.

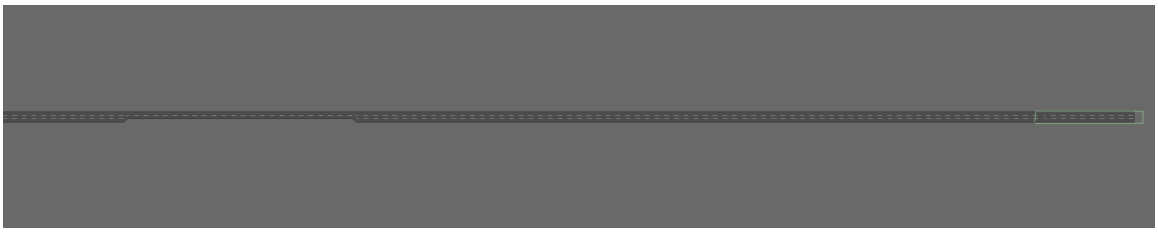

Figure 34: Aerial View of Lane-Drop Area of Network #1 in PARAMICS

4.1.5.1.2 Network #2: Hypothetical Freeway Segment with On/Off Ramps & a Dedicated Lane

Network #2 consists of a 6 km straight stretch of a hypothetical freeway segment with 2 pairs of on and off ramps located at the 2 km mark and the 4 km mark, and is pictured in Figure 35 below. A more detailed view of one of the on/off ramp pairs is shown in Figure 36. The freeway segment consists of 4 lanes with the left-most lane set as a dedicated "eco-lane" for Eco-CACC platoons. The intent of creating network #2 was to test Eco-CACC lateral maneuvers such as platoon splitting, where a vehicle needs to leave a platoon in order to reach its desired freeway exit. In addition, network #2 was used to test heterogeneous route-based platoon formation, which occurs when vehicles with different freeway exits are permitted to join the same platoon.

Figure 35: Aerial View of Network #2 in PARAMICS

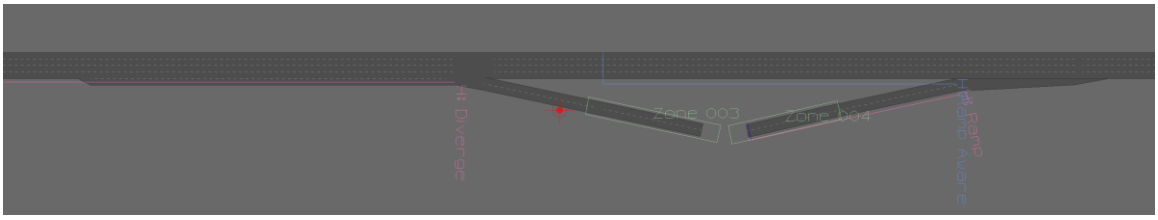

Figure 36: Aerial View of On/Off Ramp Pair in Network #2 in PARAMICS

4.1.5.1.3 Network #3: California SR-91 Eastbound Freeway with Dedicated Lane

Network #3 is a model of the California SR-91 Eastbound freeway. Specifically, a 15-mile stretch of the SR-91 between the Orange County line and Tyler Street in Riverside, California was modeled, as pictured in Figure 37 below. The interchange between the SR-91 E and the I-15 was also modeled and is shown in Figure 38. The number of lanes varied between 4 and 6, with 5 being the most common. In addition, the freeway segment includes 9 on/off ramp pairs, 13 origin zones, and 12 destination zones. The speed limit included links with 60 mph speed limits and links with 65 mph speed limits. The intent of utilizing a model of a real-world freeway was to test how Eco-CACC might perform if given a single dedicated lane on an existing real-world freeway. The left-most lane was set as a dedicated “eco-lane” for Eco-CACC platoons. Network #3 was originally coded and calibrated by Dr. Kanok Boriboonsomsin as documented in association with CE-CERT at UCR [Boriboonsomsin and Barth, 2006].

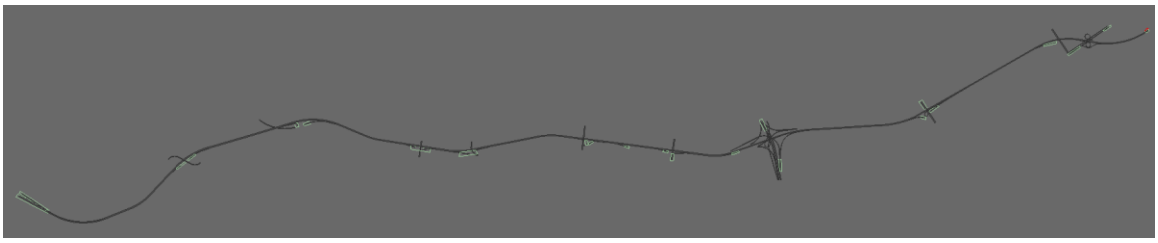

Figure 37: Aerial View of Network #3 in PARAMICS

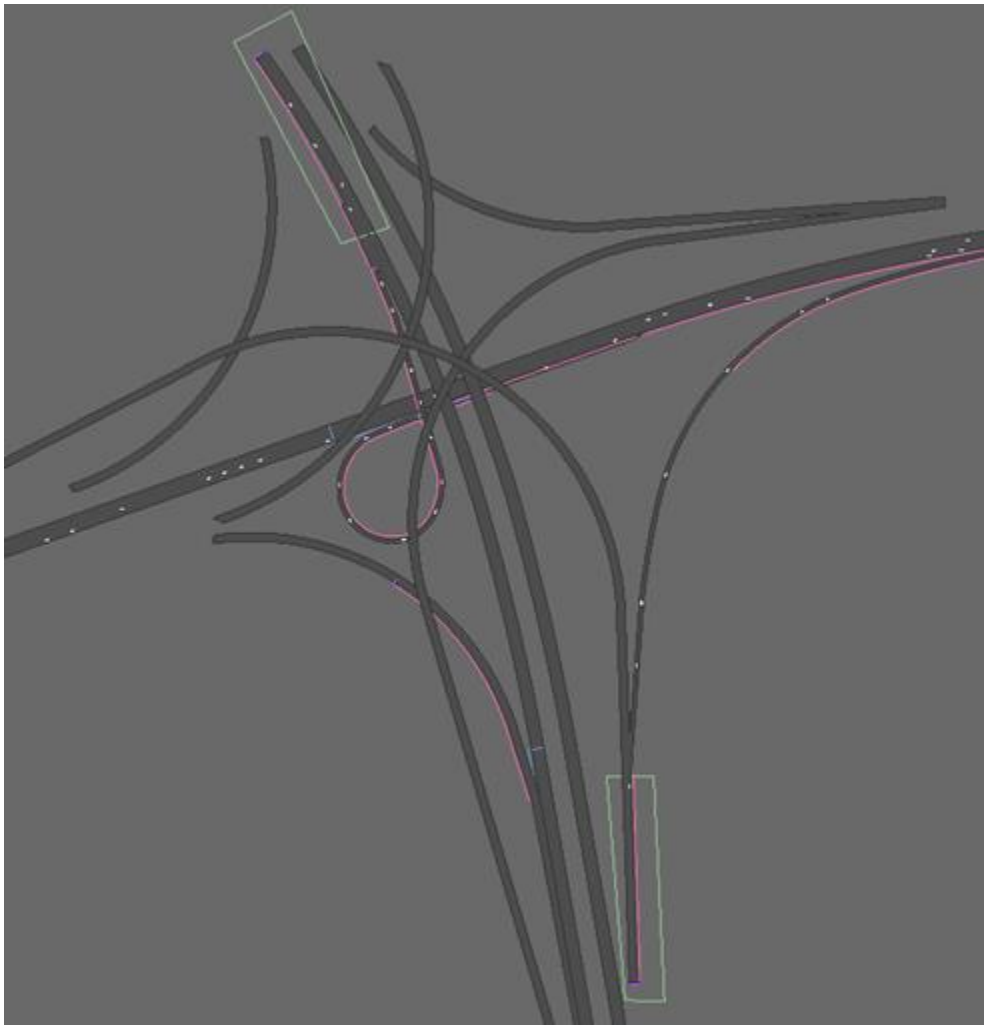

Figure 38: Aerial View of SR-91 / I-15 Interchange Model in PARAMICS

4.1.5.2 Network 1: Hypothetical Freeway Segment with Lane-Drop

For the hypothetical freeway segment with the lane-drop area, a series of four sensitivity analyses were conducted:

- Traffic volume
- Penetration rate of Eco-CACC technology
- Triggering distance
- Intra-platoon clearance.

All of the sensitivity analyses shared the following parameters: The total hourly traffic volume entering the network varied between 3,000 vehicles and 6,000 vehicles total in increments of 600. A constant demand profile was used, meaning that the vehicular demand is constant over time along the roadway. The simulation time selected was 1 hour, with additional time to allow all vehicles to enter and exit the network. A single light-duty vehicle type was used during the simulations. Unless otherwise stated, the following results are network-wide results including all vehicles across all lanes.

4.1.5.2.1 Traffic Volume Sensitivity Analysis

For the traffic volume sensitivity analysis, penetration rates of 100 percent and 0 percent were tested over the following traffic volumes: 3,000; 3,600; 4,200; 4,800; 5,400; and 6,000 vph. The triggering distance, or distance threshold when vehicles will trigger platoon-forming behavior, was set at 40 meters or less. The vehicle clearance parameter was set at 5 meters.

The traffic volume sensitivity analyses, which Figure 39 through Figure 42 and Table 11 through Table 13 show, indicate that the baseline capacity of the freeway segment was roughly 3,600 vph. After 3,600 vph, the baseline performance in terms of travel time and energy was degraded. In contrast, even at the highest volume tested, 6,000 vph, the applied Eco-CACC algorithm and merging protocol maintained the free-flow conditions present at lower volumes. Further testing at higher volumes is necessary before drawing conclusions on the limit of the capacity improvement provided by CACC. The results provided indicate that the capacity is improved by at least two-thirds. A maximum savings of more than 70 percent and 30 percent were achieved in travel time and energy, respectively. It should be noted that the significant difference in results between the baseline and the Eco-CACC algorithm are the result of the combination of the core Eco-CACC algorithm and the merging protocol. The simultaneous increase in traffic density and vehicle cooperation leads to the maintenance of free-flow traffic at higher volumes than are normally possible for today's freeways.

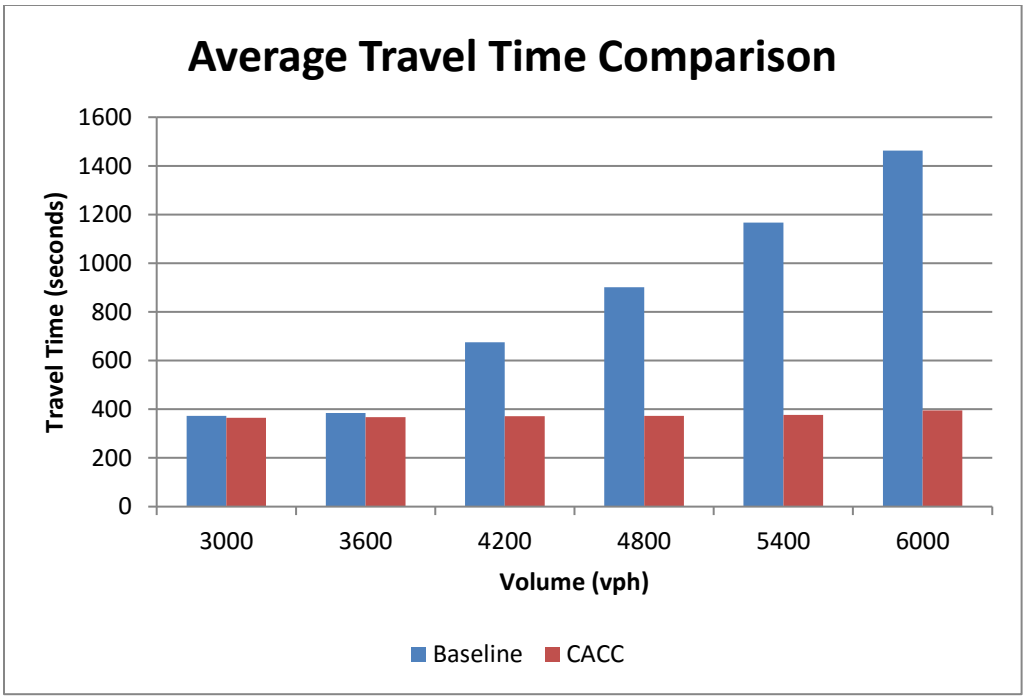

Figure 39: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates

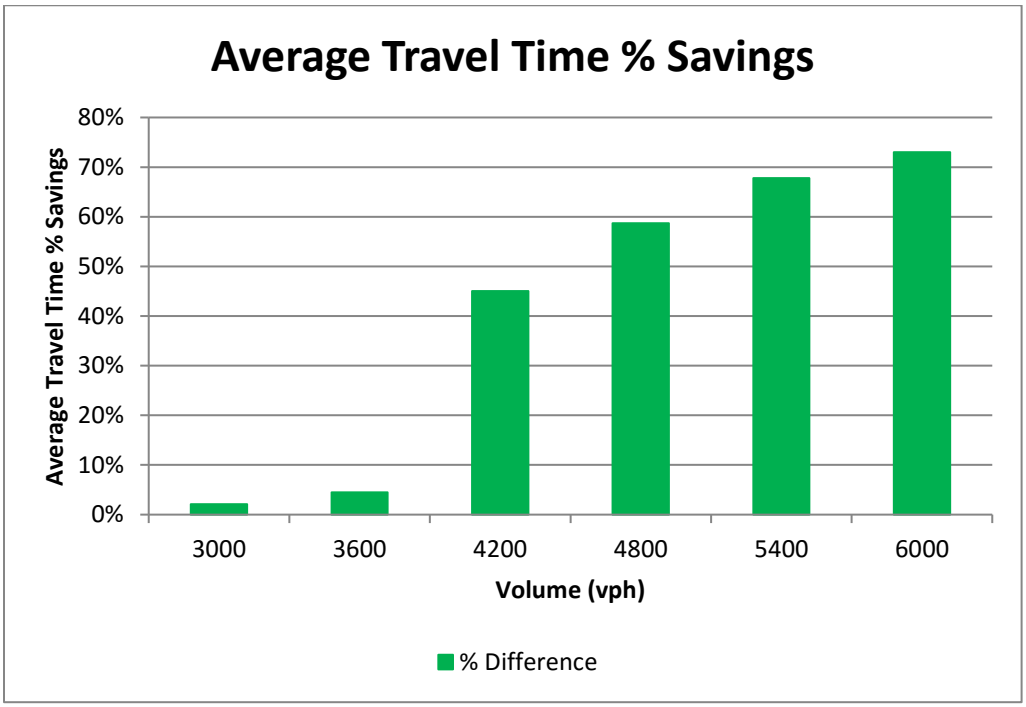

Figure 40: Average Travel Time % Savings vs. Traffic Volume for 100% vs 0% Penetration Rates

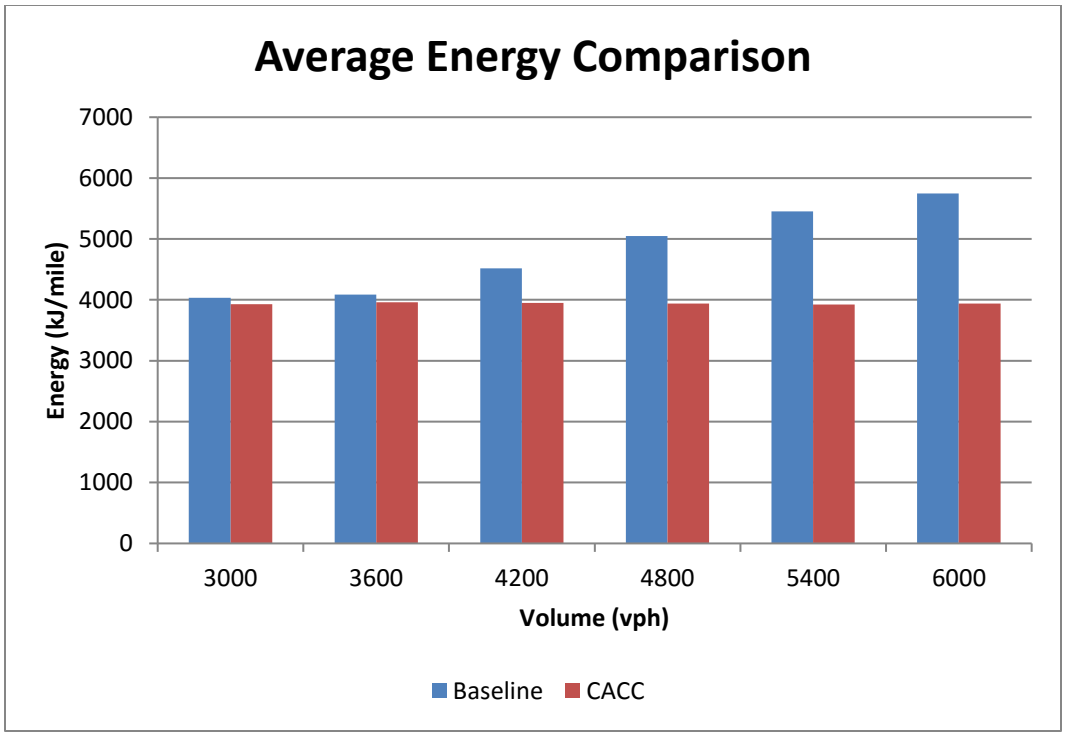

Figure 41: Energy vs. Traffic Volume for 0% & 100% Penetration Rates

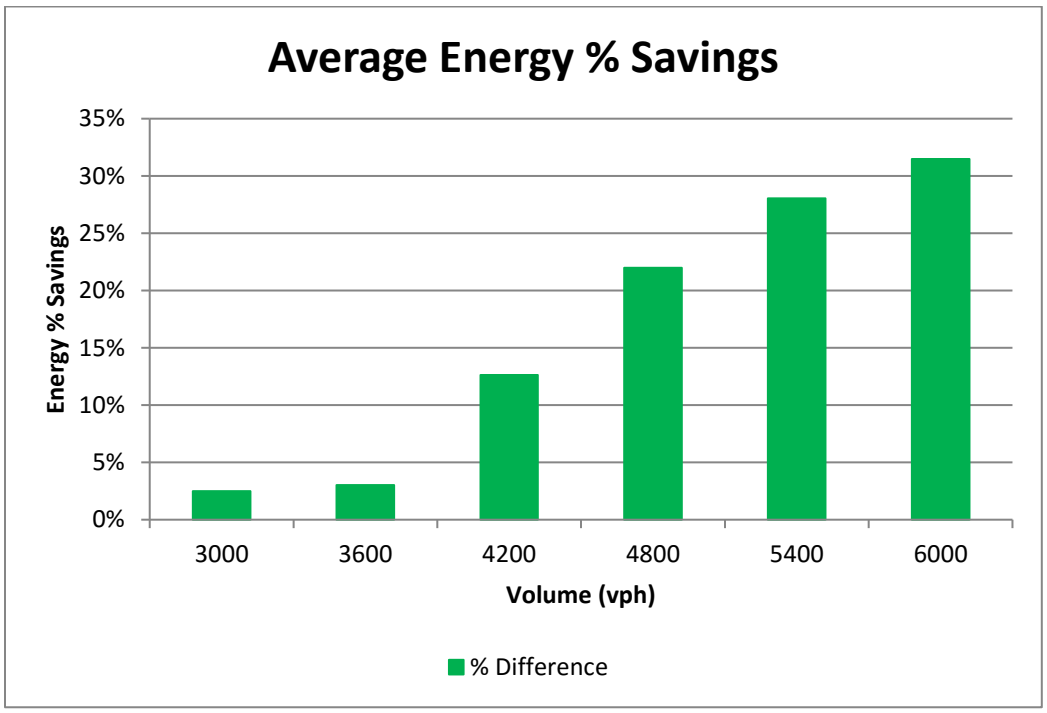

Figure 42: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates

Table 11: Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	4032.1786	289.7922	2.5689	0.1173	0.6829	0.009716	372.6657
3600	4084.5152	293.5536	2.6508	0.1208	0.6877	0.009911	384.8094
4200	4517.4448	324.6678	3.1275	0.1623	0.5977	0.008662	674.6607
4800	5048.5277	362.8366	3.5383	0.1939	0.5917	0.008679	902.0285
5400	5451.7048	391.8128	3.8299	0.2153	0.6018	0.008873	1167.4204
6000	5749.5646	413.2199	4.0465	0.2313	0.6086	0.009009	1462.9594

Table 12: Eco-CACC, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	3931.2886	282.5413	2.4089	0.1117	0.6645	0.00924	364.8120
3600	3961.6475	284.7232	2.4384	0.1127	0.6702	0.009306	367.6145
4200	3947.5324	283.7087	2.4337	0.1129	0.6644	0.009421	370.9136
4800	3938.3956	283.0520	2.4466	0.1132	0.6613	0.009452	372.7430
5400	3922.1737	281.8861	2.4518	0.1135	0.6557	0.009438	376.2061
6000	3939.0579	283.0995	2.5621	0.1157	0.6582	0.009903	395.0078

Table 13: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	2.50%	2.50%	6.23%	4.78%	2.70%	4.90%	2.11%
3600	3.01%	3.01%	8.01%	6.72%	2.54%	6.10%	4.47%
4200	12.62%	12.62%	22.18%	30.46%	-11.16%	-8.76%	45.02%
4800	21.99%	21.99%	30.85%	41.64%	-11.76%	-8.91%	58.68%
5400	28.06%	28.06%	35.98%	47.28%	-8.95%	-6.37%	67.77%
6000	31.49%	31.49%	36.68%	49.99%	-8.15%	-9.92%	73.00%

4.1.5.2.2 Eco-CACC Penetration Rate Sensitivity Analysis

A vehicle with Eco-CACC technology is assumed to have sensors, controllers, and CV technology capable of communicating and receiving information from surrounding vehicles and infrastructure. A total of seven non-zero penetration rates of Eco-CACC technology were simulated at the aforementioned range of traffic volumes. The penetration rates tested were 5 percent, 10%, 20%, 40%, 60%, 80%, and 100%. The triggering distance was set at 40 meters and the vehicle clearance parameter was set at 5 meters.

The penetration rate of Eco-CACC technology sensitivity analyses, which Figures 43-44 and Tables 14-15 show, indicate that the higher the penetration rate, the greater the benefits. The benefits in capacity improvement appear to be positively correlated with the penetration rate. The higher the penetration rate, the higher the capacity, and vice versa. Furthermore, the results also appear to indicate a lower and upper threshold for penetration rates in terms of performance. Penetration rates less than or equal to 10% have little benefit as a result of the sparsity of platoons. However, penetration rates greater than or equal to 60% have similar performance (within the range of tested volumes).

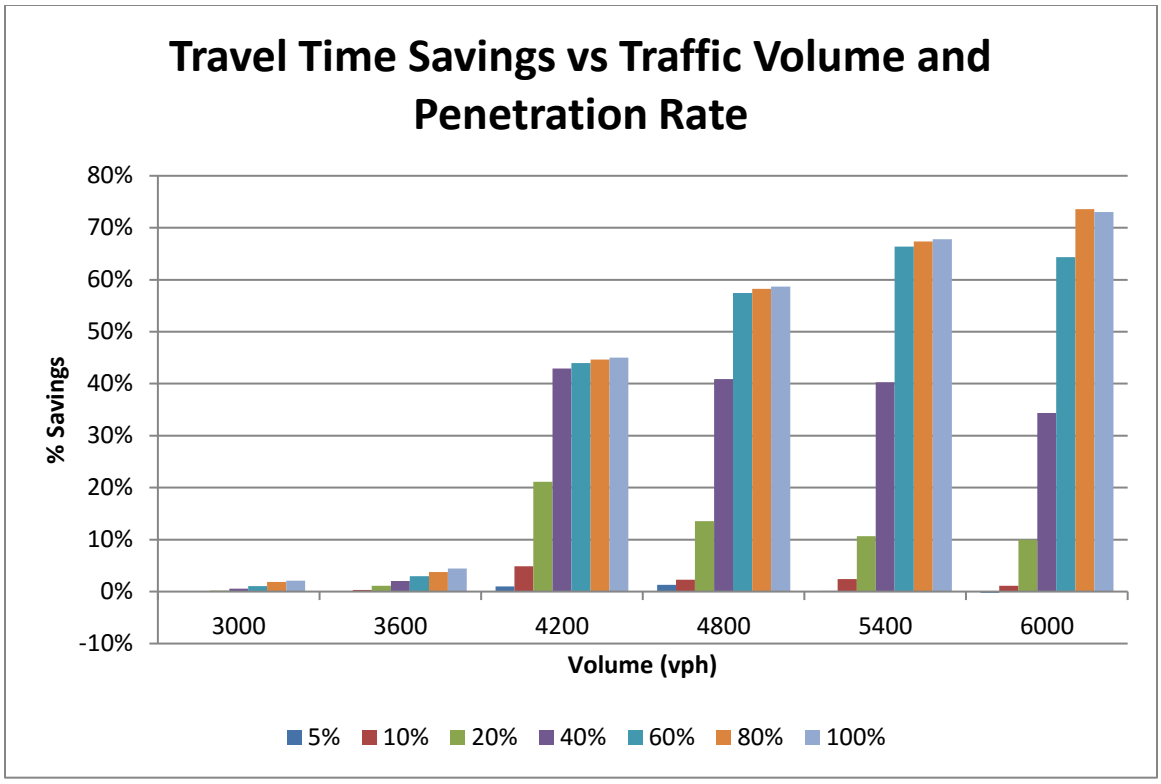

Figure 43: Average Travel Time % Savings vs. Traffic Volume and Penetration Rate

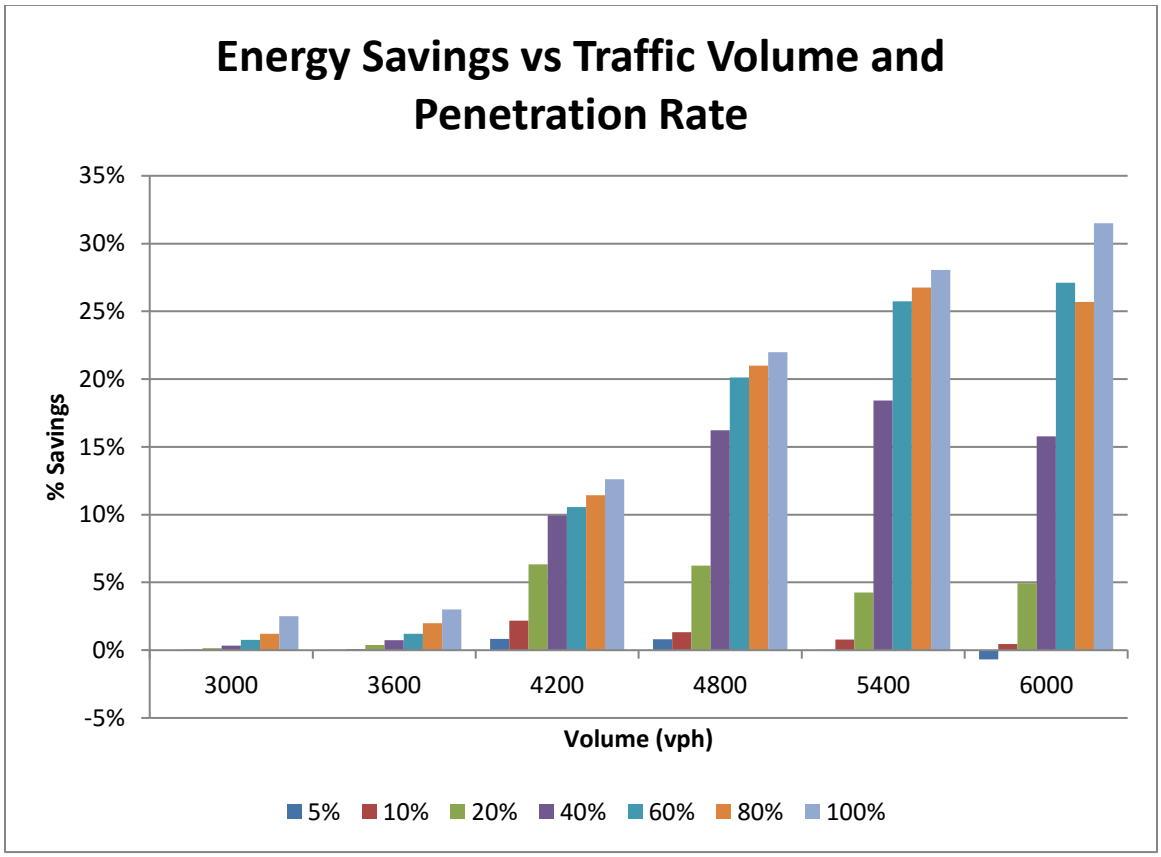

Figure 44: Energy % Savings vs. Traffic Volume and Penetration Rate

Table 14: Travel Time % Improvement of Eco-CACC over Baseline, Penetration Rate Sensitivity Analysis Results

Volume (vph)	*5%*	*10%*	*20%*	*40%*	*60%*	*80%*	*100%*
3000	0.04%	-0.06%	0.18%	0.59%	1.03%	1.83%	2.11%
3600	0.10%	0.30%	1.10%	2.06%	2.95%	3.77%	4.47%
4200	0.96%	4.90%	21.11%	42.94%	43.96%	44.64%	45.02%
4800	1.31%	2.25%	13.53%	40.92%	57.45%	58.24%	58.68%
5400	0.14%	2.38%	10.64%	40.26%	66.35%	67.38%	67.77%
6000	-0.21%	1.11%	9.96%	34.37%	64.32%	73.57%	73.00%

Table 15: Energy % Improvement of Eco-CACC over Baseline, Penetration Rate Sensitivity Analysis Results

Volume (vph)	5%	10%	20%	40%	60%	80%	100%
3000	-0.02%	0.06%	0.15%	0.33%	0.77%	1.20%	2.50%
3600	0.00%	0.07%	0.37%	0.74%	1.22%	1.98%	3.01%
4200	0.84%	2.17%	6.34%	9.95%	10.56%	11.44%	12.62%
4800	0.81%	1.33%	6.25%	16.23%	20.12%	20.99%	21.99%
5400	0.01%	0.78%	4.26%	18.41%	25.74%	26.76%	28.06%
6000	-0.68%	0.46%	4.95%	15.78%	27.11%	25.69%	31.49%

4.1.5.2.3 Triggering Distance Sensitivity Analysis

Triggering distance is the inter-vehicular clearance at which a vehicle will attempt to join a platoon ahead. Three triggering distances (20 meters, 30 meters, and 40 meters) were simulated at the aforementioned volumes. The vehicle clearance parameter was set at 5 meters and the penetration rate of Eco-CACC technology was set to 100 percent.

The triggering distance sensitivity analyses, which Figures 45-46 show, indicate that there are only small improvements in increasing triggering distance. Recall that the triggering distance is the inter-vehicular distance at which an Eco-CACC–equipped vehicle will elect to begin closely following the vehicle ahead. In a real-life scenario, this parameter may be set manually by a driver. Depending on the ratio of triggering distance to a driver's normal car-following distance, a driver may need to drive closer to a vehicle to trigger platoon formation. In simulation, the observed car-following distance was roughly 25 meters. As a result, when the triggering distance was set to 20 meters, very few platoons formed before the lane-drop area. As a result of the disturbance of vehicles changing lanes in the lane-drop area, approaching vehicles slowed down, leading to car-following distances of less than 20 meters, and thereby triggering platoon formation. In

contrast, the selection of triggering distances greater than 20 meters, such as 30 meters and 40 meters, led to platoon formation immediately after vehicle insertion into the network. These smaller platoons increased in size when passing through the lane-drop area. In summary, increasing the triggering distance influenced how early a vehicle joined a platoon, the overall length of platoons, and, to a small degree, the number of vehicles involved in platooning behavior. Therefore, in spite of the apparently small difference in benefits between different triggering distances in Tables 16-17, the underlying fundamental vehicle behavior is significantly different among the tested triggering distances in simulation.

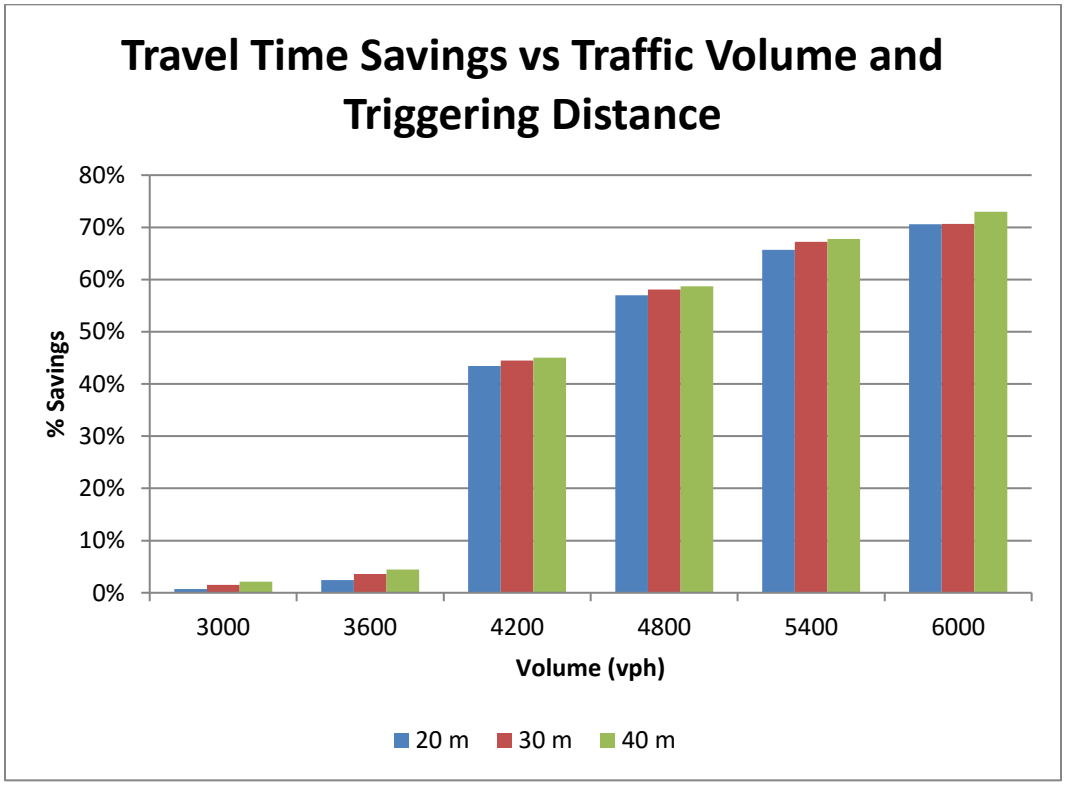

Figure 45: Average Travel Time % Savings vs. Traffic Volume and Triggering Distance

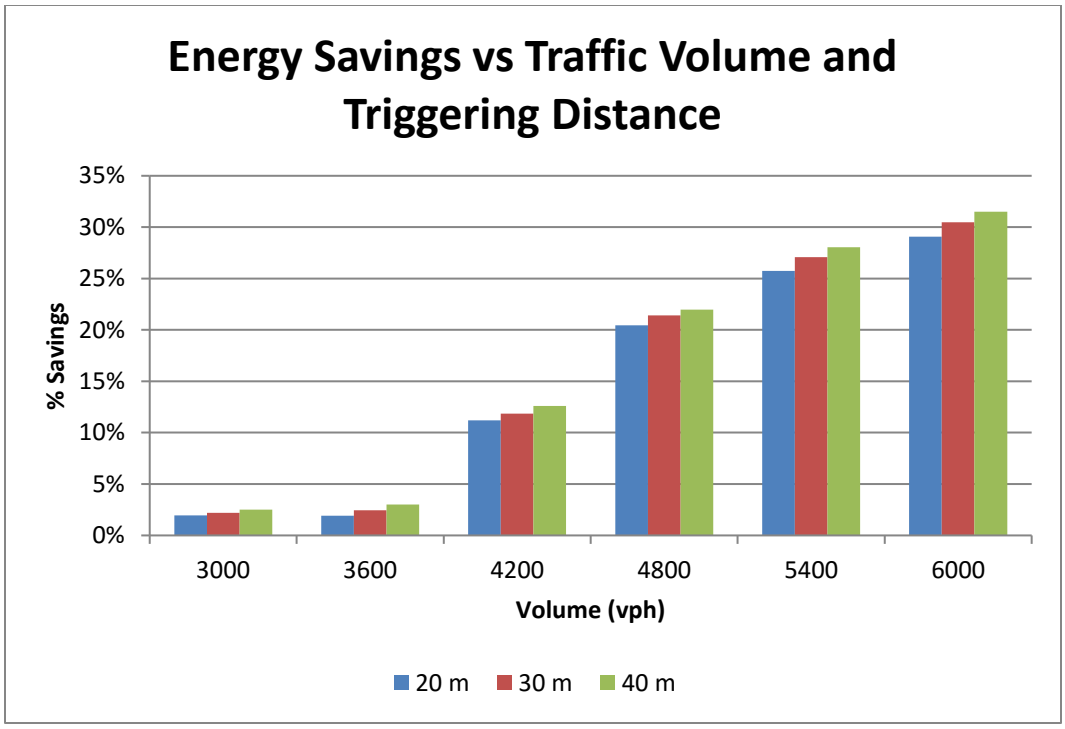

Figure 46: Energy % Savings vs. Traffic Volume and Triggering Distance

Table 16: Travel Time % Improvement of Eco-CACC over Baseline, Triggering Distance Sensitivity Analysis Results

Volume (vph)	*20 m*	*30 m*	*40 m*
3000	0.73%	1.51%	2.11%
3600	2.42%	3.60%	4.47%
4200	43.45%	44.45%	45.02%
4800	56.99%	58.10%	58.68%
5400	65.67%	67.22%	67.77%
6000	70.62%	70.69%	73.00%

Table 17: Energy % Improvement of Eco-CACC over Baseline, Triggering Distance Sensitivity Analysis Results

Volume (vph)	20 m	30 m	40 m
3000	1.95%	2.21%	2.50%
3600	1.92%	2.44%	3.01%
4200	11.20%	11.85%	12.62%
4800	20.45%	21.42%	21.99%
5400	25.74%	27.08%	28.06%
6000	29.08%	30.47%	31.49%

4.1.5.2.4 Intra-Platoon Spacing Sensitivity Analysis

Intra-platoon clearance is the clearance between vehicles travelling in the same platoon. Intra-platoon clearances of 5 meters, 10 meters, 15 meters, and 20 meters were simulated at each of the aforementioned volumes. A triggering distance of 40 meters was selected along with a penetration rate of Eco-CACC technology of 100%.

The intra-platoon clearance sensitivity analyses, which Figures 47-48 and Tables 18-19 show, indicate that there is very little performance drop as the intra-platoon clearance is increased. However, increasing the intra-platoon clearance does lower the achievable traffic density, therefore lowering the achievable capacity. Increasing intra-platoon clearance led to increases in average platoon size and length (the number of vehicles within a platoon and the distance between the leader and the tail of the platoon, respectively). To fully observe the effects of platoon growth, no restraints were placed on the maximum platoon length. Future work should consider maximum platoon lengths. The case of 15-meter intra-platoon clearance corresponds roughly to a 0.6-second gap. One of the advantages afforded by larger gaps is that CACC controllers are permitted

more time to respond to emergency situations. In conclusion, there is a slight tradeoff between overall capacity improvement and level of safety.

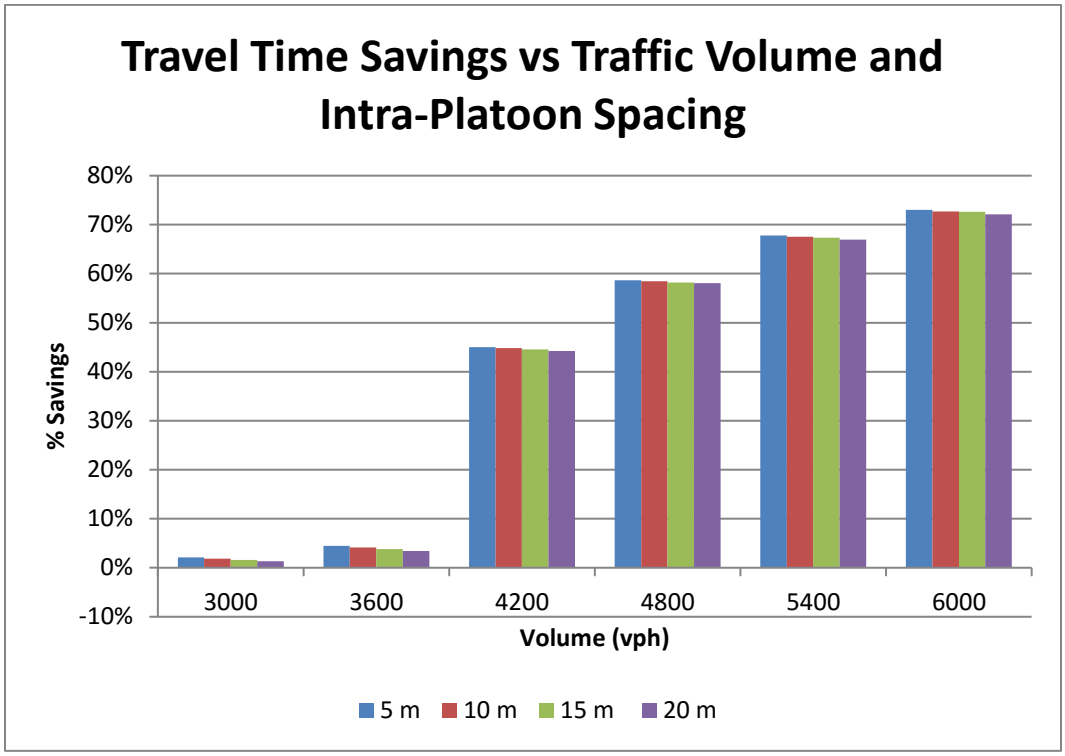

Figure 47: Average Travel Time % Savings vs. Traffic Volume and Intra-Platoon Spacing

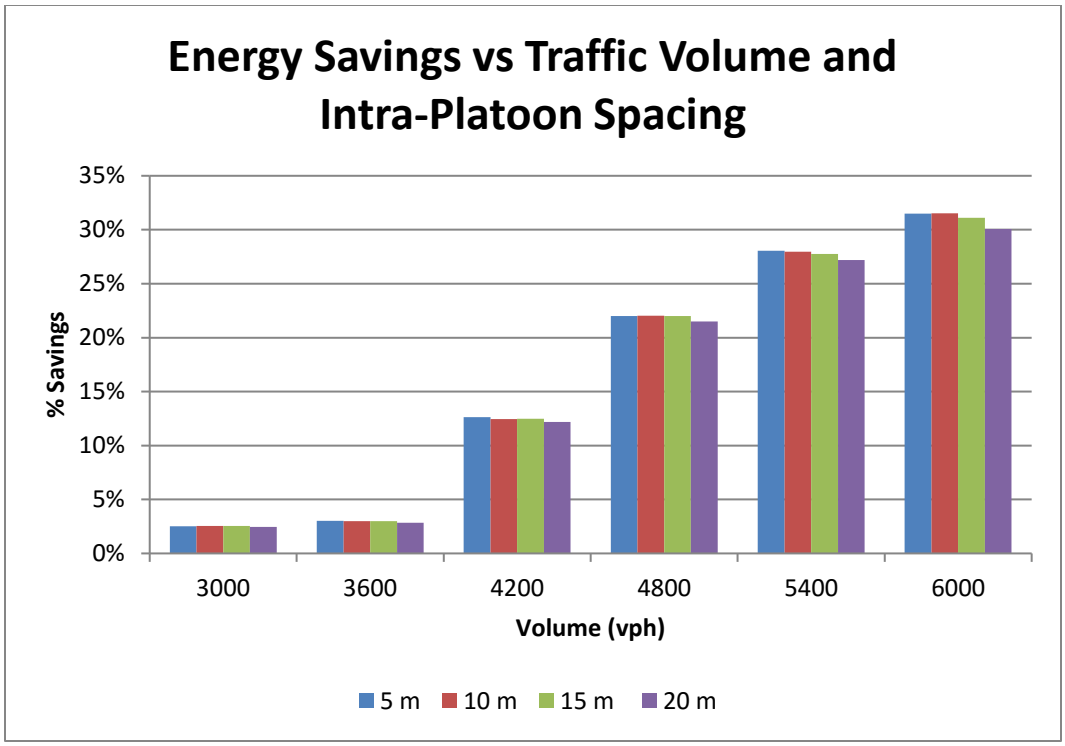

Figure 48: Energy % Savings vs. Traffic Volume and Intra-Platoon Spacing

Table 18: Travel Time % Improvement of Eco-CACC over Baseline, Intra-Platoon Spacing Sensitivity Analysis Results

Volume (vph)	*5 m*	*10 m*	*15 m*	*20 m*
3000	2.11%	1.85%	1.60%	1.33%
3600	4.47%	4.11%	3.81%	3.43%
4200	45.02%	44.80%	44.56%	44.23%
4800	58.68%	58.44%	58.21%	58.07%
5400	67.77%	67.52%	67.31%	66.94%
6000	73.00%	72.69%	72.65%	72.11%

Table 19: Energy % Improvement of Eco-CACC over Baseline, Intra-Platoon Spacing Sensitivity Analysis Results

Volume (vph)	5 m	10 m	15 m	20 m
3000	2.50%	2.55%	2.53%	2.47%
3600	3.01%	3.00%	3.00%	2.84%
4200	12.62%	12.46%	12.49%	12.17%
4800	21.99%	22.04%	22.00%	21.50%
5400	28.06%	27.96%	27.76%	27.18%
6000	31.49%	31.53%	31.11%	30.06%

4.1.5.3 Network 2: Hypothetical Freeway Segment with On-Ramps/Off-Ramps and a Dedicated Lane ("Eco-Lane")

For the hypothetical freeway segment with on-ramps/off-ramps, a sensitivity analysis for traffic volume was conducted. The total hourly traffic volume entering the network varied between 3,000 vehicles and 7,800 vehicles total in increments of 600. A constant demand profile was used. The simulation time selected was 1 hour, with additional time to allow all vehicles to enter and exit the network. A single vehicle type, in this case a single occupant vehicle, was used during the simulations. The left-most lane was set as a dedicated "eco-lane" for Eco-CACC platoons. This "eco-lane" was designed to operate much like an HOV lane but restricted to "eco-vehicles" rather than HOVs. Vehicles in platoons that are close to their desired exit initiate the platoon-splitting protocol. The platoon-merging protocol is not applied to the network because of the absence of lane-drops. In addition to the traffic volume sensitivity analysis, an analysis of network savings versus dedicated lane savings is provided. Finally, a section on the benefits of mainstream vehicles choosing the dedicated lane is included. Unless

otherwise stated, the following results are network-wide results including all vehicles across all lanes.

4.1.5.3.1 Traffic Volume Sensitivity Analysis

For the traffic volume sensitivity analysis, a penetration rate of 100% and 0% were tested over the following traffic volumes: 3000, 3600, 4200, 4800, 5400, 6000, 6600, 7200, and 7800 vehicles per hour (vph). The triggering distance, or distance threshold when vehicles will trigger platoon forming behavior, was set at 40 meters. The vehicle spacing parameter was set at 5 meters.

The traffic volume sensitivity analyses for network 2, shown in Figures 49-52 and Tables 20-22, indicate that there is only a small improvement in terms of travel time and energy at volumes less than or equal to 6600 vph. There is a significant improvement at volumes greater than 6600 vph, around 28% for travel time and 9% for energy at 7800 vph. Even the presence of a single dedicated lane led to an increase in overall network capacity. The roadway sections immediately prior to on-ramps led to platoon formation as vehicles would change lanes to the left to better accommodate vehicles entering the freeway. Benefits for volumes less than 6600 vph were 0-2% for travel time and 0-4% for energy.

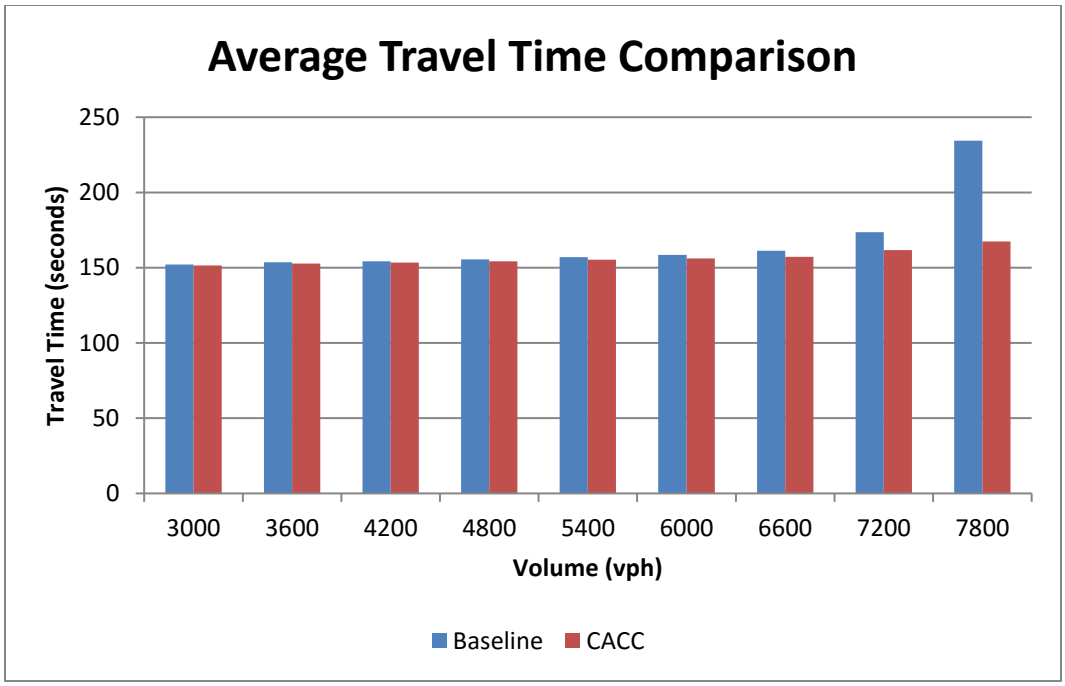

Figure 49: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates

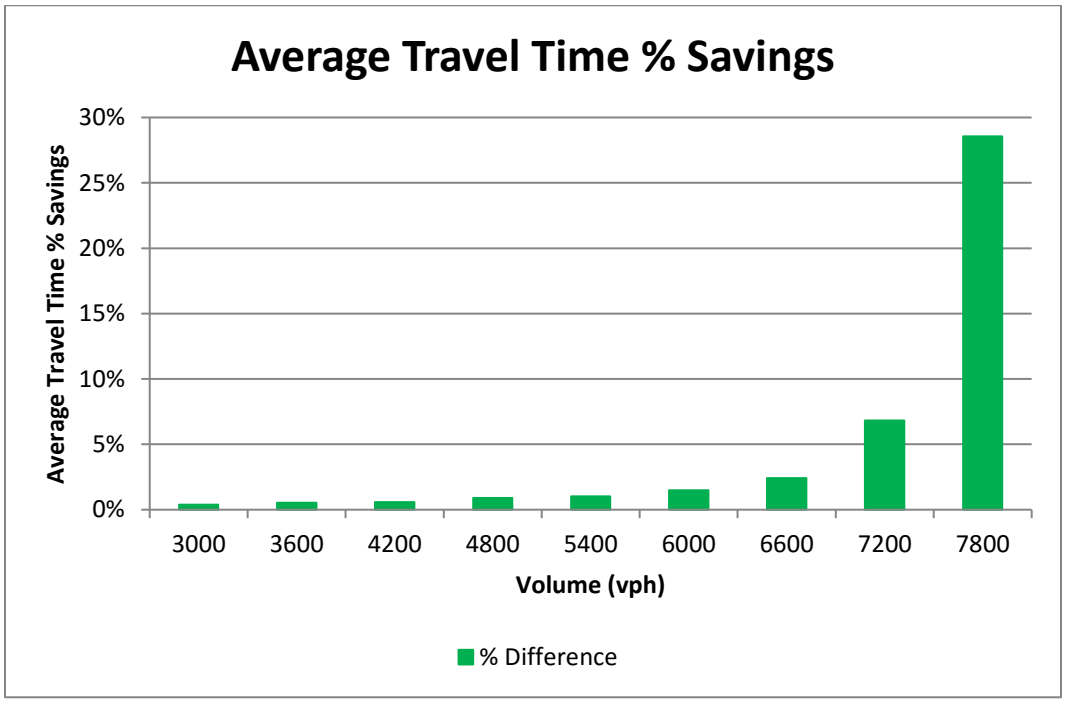

Figure 50: Average Travel Time % Savings vs. Traffic Volume for 100% vs 0% Penetration Rates

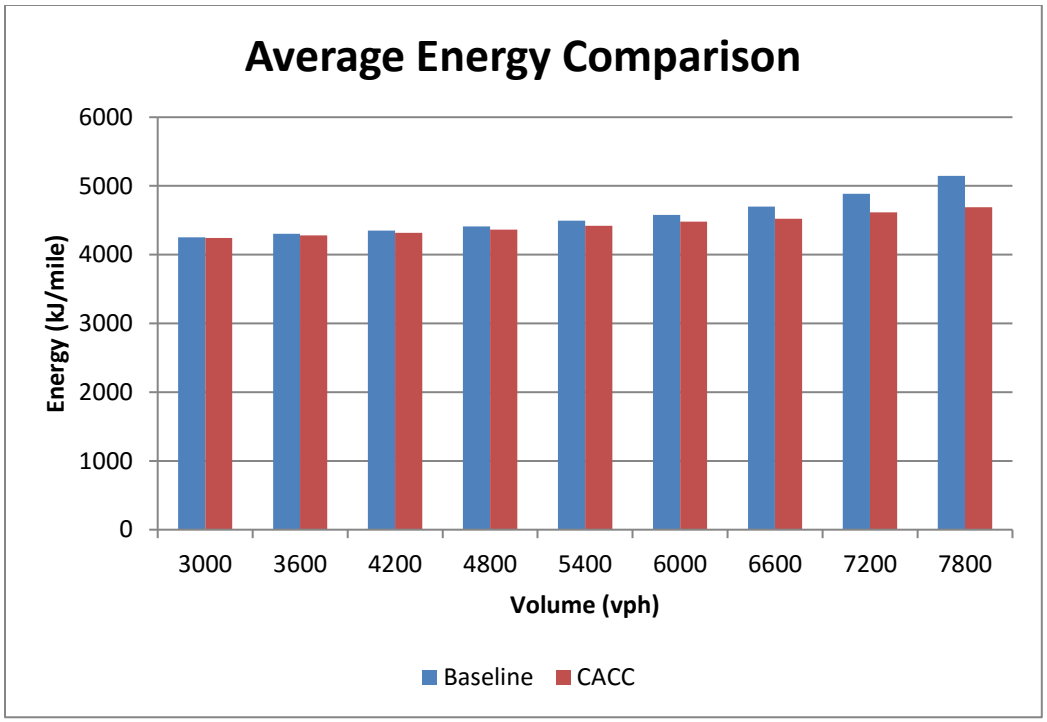

Figure 51: Energy vs. Traffic Volume for 0% & 100% Penetration Rates

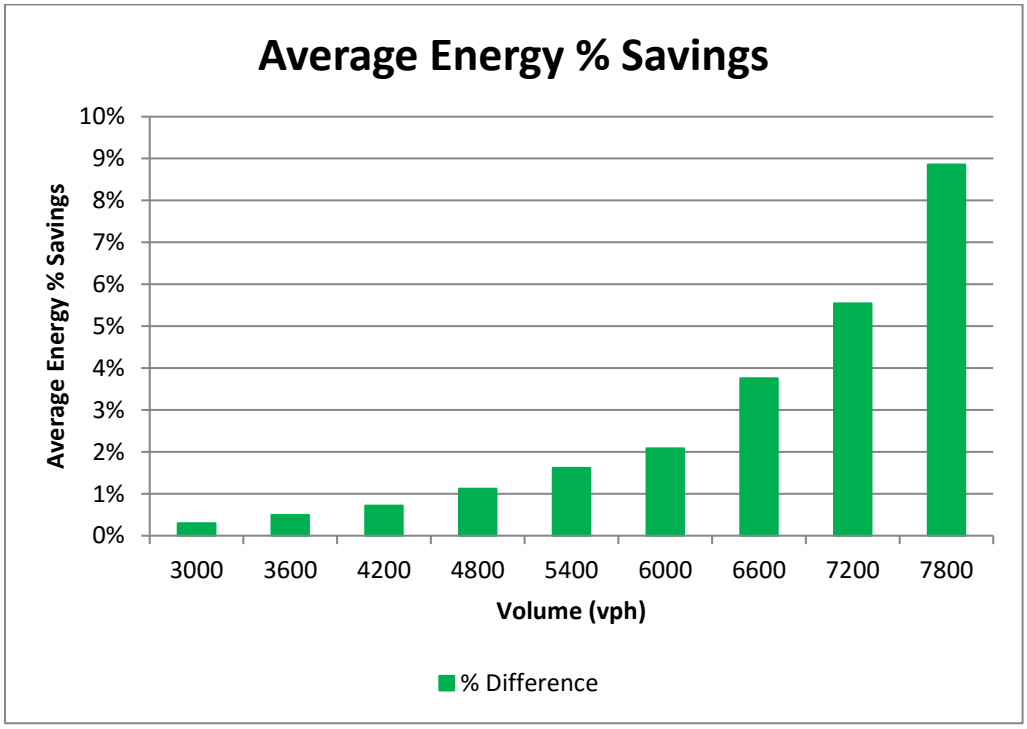

Figure 52: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates

Table 20: Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	4254.1986	305.7488	3.5359	0.1317	0.7489	0.013017	152.0459
3600	4304.3954	309.3564	3.6912	0.1350	0.7606	0.013705	153.5413
4200	4351.4544	312.7386	3.8424	0.1378	0.7719	0.014304	154.3315
4800	4411.6344	317.0637	4.0727	0.1425	0.7850	0.015279	155.5337
5400	4493.7584	322.9659	4.3290	0.1479	0.8035	0.016317	156.9623
6000	4578.3498	329.0454	4.5998	0.1537	0.8219	0.017685	158.5112
6600	4697.7769	337.6286	4.9820	0.1619	0.8479	0.019147	161.1927
7200	4884.9340	351.0795	5.4530	0.1748	0.8814	0.021785	173.5729
7800	5147.9073	369.9792	5.7445	0.1926	0.9001	0.022762	234.4004

Table 21: Eco-CACC, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	4241.4650	304.8336	3.4885	0.1307	0.7470	0.012837	151.4717
3600	4283.0063	307.8192	3.6275	0.1335	0.7571	0.013427	152.7292
4200	4320.1996	310.4923	3.7181	0.1353	0.7660	0.013876	153.4472
4800	4362.2907	313.5174	3.8883	0.1387	0.7758	0.014526	154.1440
5400	4421.0574	317.7409	4.0916	0.1429	0.7888	0.015451	155.3654
6000	4482.8501	322.1819	4.2785	0.1469	0.8026	0.016254	156.1719
6600	4521.3755	324.9507	4.4321	0.1498	0.8116	0.016848	157.2837
7200	4614.2537	331.6259	4.6860	0.1556	0.8311	0.018004	161.7236
7800	4691.8205	337.2005	4.9165	0.1612	0.8468	0.019236	167.4706

Table 22: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
3000	0.30%	0.30%	1.34%	0.79%	0.25%	1.38%	0.38%
3600	0.50%	0.50%	1.72%	1.07%	0.46%	2.03%	0.53%
4200	0.72%	0.72%	3.23%	1.82%	0.75%	2.99%	0.57%
4800	1.12%	1.12%	4.53%	2.69%	1.18%	4.93%	0.89%
5400	1.62%	1.62%	5.48%	3.36%	1.83%	5.31%	1.02%
6000	2.09%	2.09%	6.99%	4.46%	2.35%	8.09%	1.48%
6600	3.75%	3.75%	11.04%	7.43%	4.29%	12.01%	2.43%
7200	5.54%	5.54%	14.07%	10.97%	5.71%	17.36%	6.83%
7800	8.86%	8.86%	14.41%	16.29%	5.92%	15.49%	28.55%

4.1.5.3.2 Traffic Network Savings vs. Dedicated Lane Savings

The average overall energy savings considering the entire network (Figure 53) differs from the average energy savings experienced in the dedicated lane. Energy data for the dedicated lane was defined as the aggregation of energy data for vehicles while they were on the dedicated lane. Therefore, if a vehicle changes lanes into the dedicated lane, energy data is accumulated as long as the vehicle remains in the dedicated lane. If a vehicle changes lanes from the dedicated lane, then that vehicle no longer contributes energy data to the dedicated lane analysis. Note that vehicles with different routes use the dedicated lane. Furthermore, based on the aforementioned definition, there is no corresponding dedicated lane travel time definition due to lane changes and diverse routes. In addition to showing the dedicated lane savings, the average savings for the remaining three lanes are also shown in Figure 53. Figure 54 shows the relative benefit of choosing the dedicated lane over a non-dedicated lane.

The additional analysis of benefits shown in Figures 53-54 provides insight into the operation of the algorithms applied to the network. At relatively low traffic volumes, the vehicles on the dedicated lane expended slightly more energy than vehicles in the baseline scenario due to acceleration during platoon formation. The average overall network energy savings were around 0% due to the balance of a 4% cost in energy on the dedicated lane and a 1% reduction in energy on the remaining lanes. At higher volumes, the dedicated lane energy savings increased to a maximum of 14%. In addition, the indirect benefit of non-dedicated lane energy savings reached roughly 4% at the highest volume. Therefore, the benefits in preserving relatively free-flow traffic outweigh the small energy costs associated with conducting automated maneuvers.

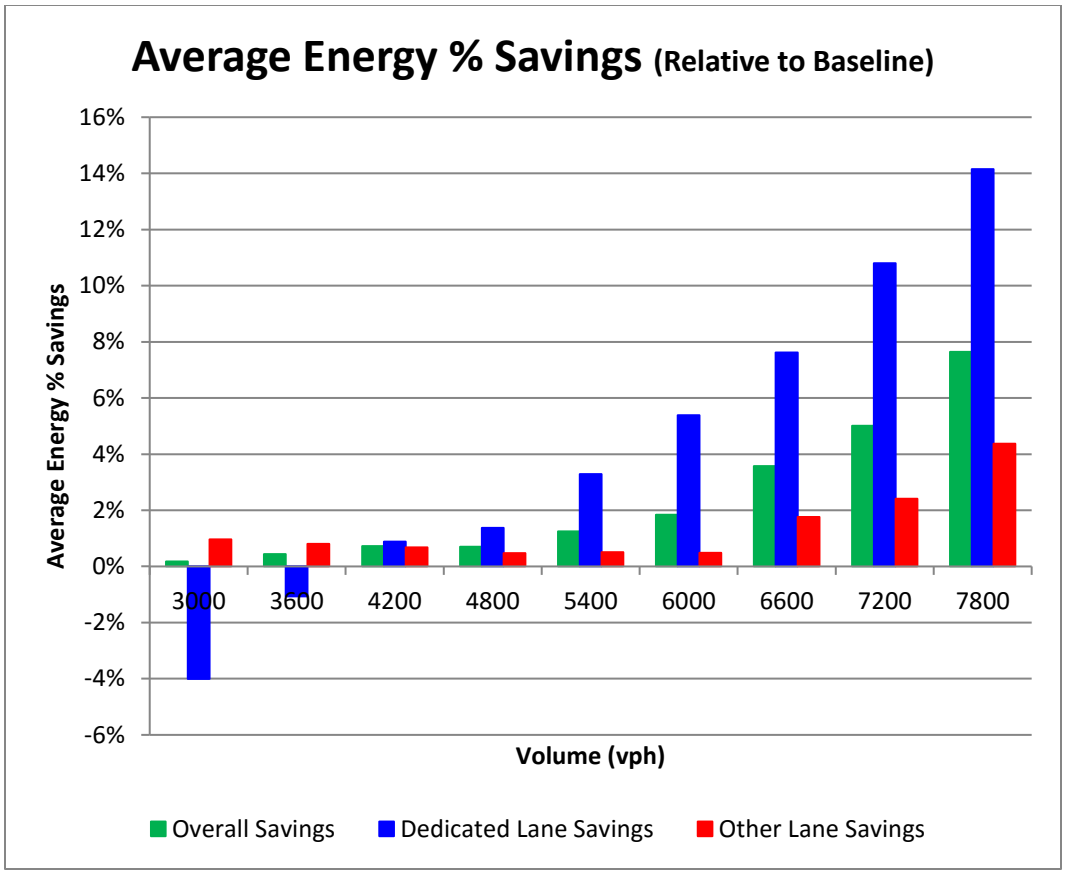

Figure 53: Dedicated Lane Energy Savings vs. Traffic Volume

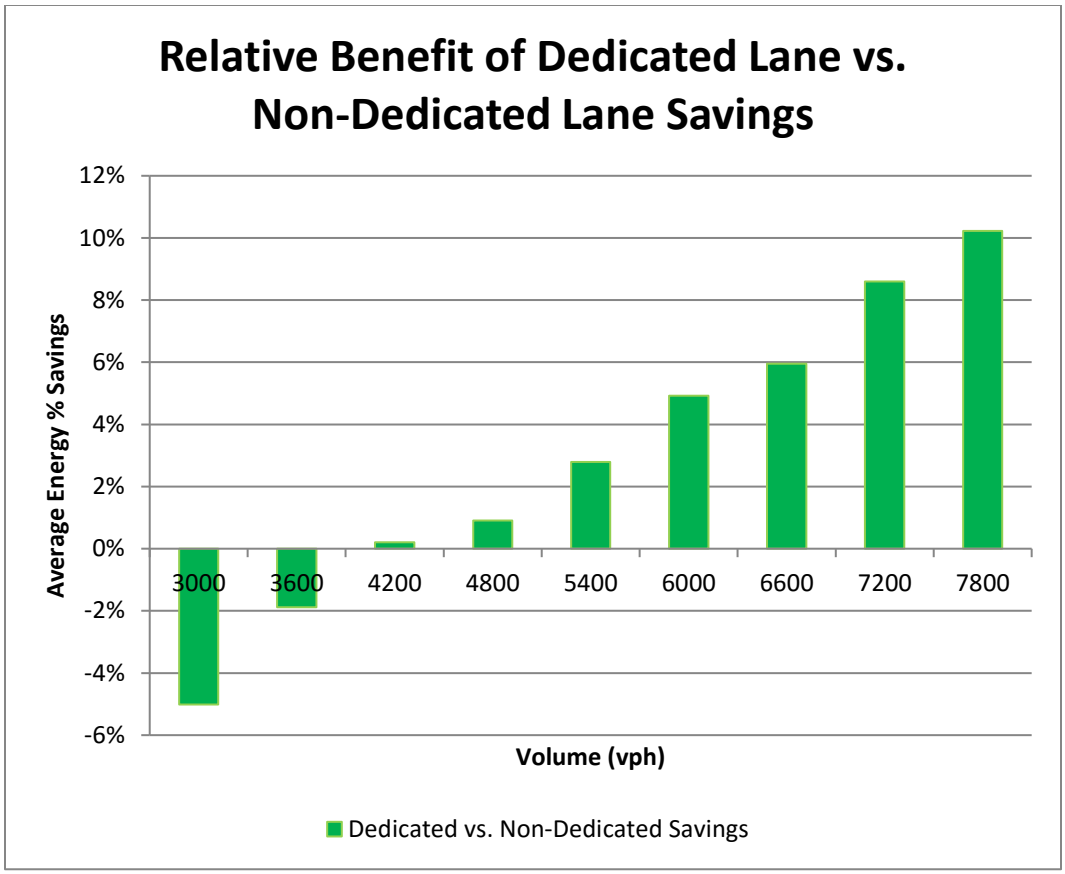

Figure 54: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings

4.1.5.3.3 Benefits of Mainstream Vehicles Choosing the Dedicated Lane

In addition to aggregating data by lane, data may also be aggregated for vehicles following a specific route. The selected route which made the most use of the dedicated lane was the mainstream route. The mainstream route was defined as the only route not including on or off ramps. Along the mainstream route, data collection was further divided among vehicles which chose to use the dedicated lane, and vehicles which did not choose the dedicated lane. The relative travel time and energy benefits of choosing the dedicated lane versus a non-dedicated lane is shown in Figure 55. Part of the reason

why energy savings are negative at most volumes is due to the additional energy required for vehicles to change lanes into the dedicated lane due to congestion on non-dedicated lanes. In order to isolate the benefits vehicles obtain in the dedicated lane an additional experiment was conducted in which the vehicles which started and remained in the dedicated lane were compared with all other vehicles with the same route. These results are shown in Figure 56.

The apparent discrepancy in energy savings between Figures 53 and 55 may be attributed to Figure 55 including vehicles expending energy changing lanes into the dedicated lane due to congestion on non-dedicated lanes. Isolating for vehicles which start and end in the dedicated lane reveals that these individual vehicles obtain a 6-10% savings in travel time and a -2-5% savings in energy relative to other vehicles along the same route (Figure 56). The small negative energy savings at lower traffic volumes are due to additional energy being expended to conduct automated maneuvers. At higher traffic volumes, the energy rate increases more quickly for non-dedicated lanes than the dedicated lane, thereby outweighing the energy costs of automated maneuvering. A visual comparison of Figures 55 and 56 reveals that vehicles which start in the dedicated lane have a benefit over vehicles which wait to change lanes into the dedicated lane due to encountering congestion.

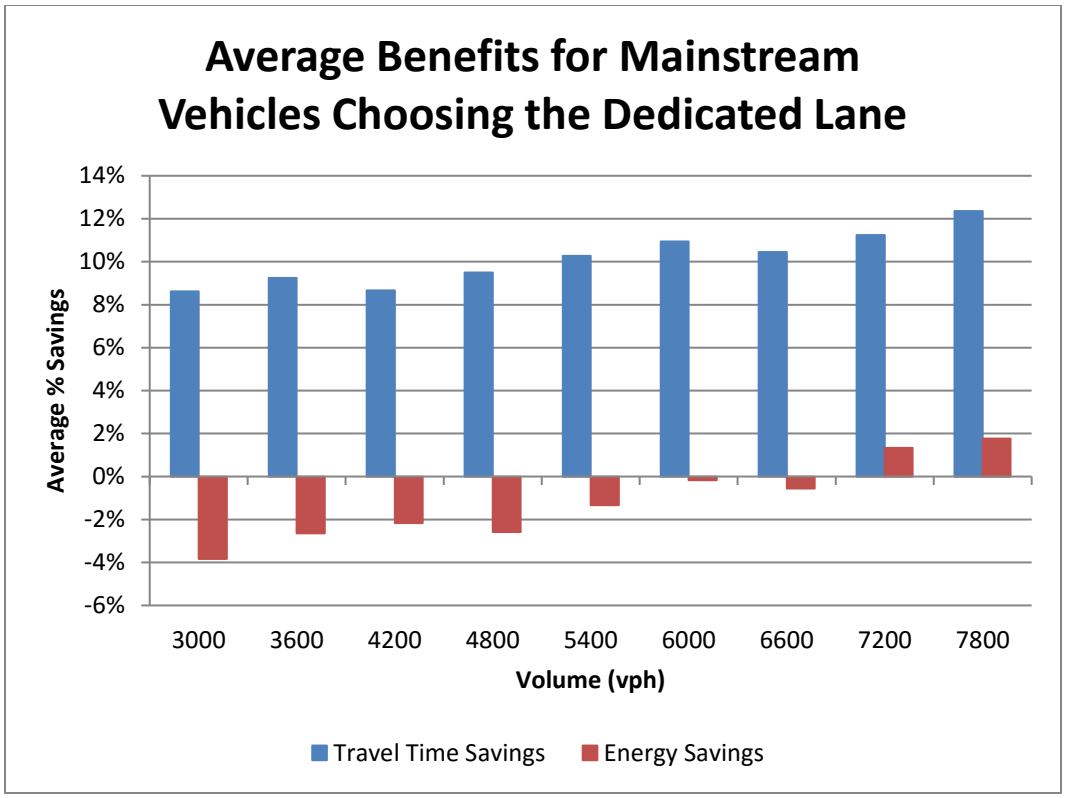

Figure 55: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane

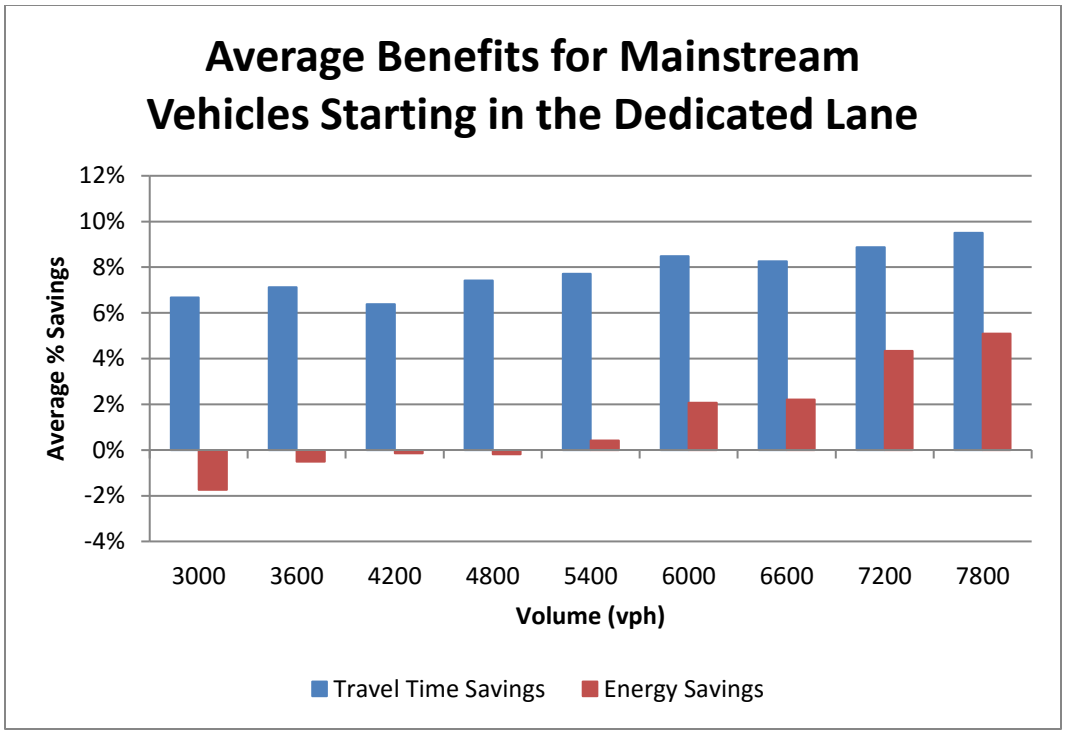

Figure 56: Average Benefits for Mainstream Vehicles Starting in the Dedicated Lane

4.1.5.4 Network 3: SR-91 E Freeway with Dedicated Lane ("Eco-Lane")

For the SR-91 E freeway, a sensitivity analysis for traffic volume was conducted. The total hourly traffic volume entering the network was varied between 25,000 vehicles and 37,000 vehicles total in increments of 3,000. A constant demand profile was used. The simulation time selected was 2 hours, with separate profiles used for each hour. The case of 25,000 vehicles was based on calibration of real-world data. A single vehicle type was used during the simulations. The left-most lane was set as a dedicated lane for Eco-CACC platoons. The number of mainstream lanes varies between 4 and 6. Vehicles in platoons that are close to their desired exit initiate the platoon-splitting protocol. The platoon-merging protocol is not applied to the network because of the absence of lane-

drops. Unless otherwise stated, the following results are network-wide results including all vehicles across all lanes.

4.1.5.4.1 Traffic Volume Sensitivity Analysis

For the traffic volume sensitivity analysis, penetration rates of 100% and 0% were tested over the following traffic volumes: 25,000; 28,000; 31,000; 34,000; and 37,000 vehicles. The triggering distance, or distance threshold when vehicles will trigger platoon-forming behavior, was set at 40 meters. Two values were used for the vehicle clearance parameter. The vehicle clearance parameter was first set at 5 meters to assess the maximum possible benefit of applying CACC to a real-world traffic network. Next, the vehicle clearance parameter was set to 15 meters to reflect a realistic deployment scenario.

As shown in Figures 57-70 and Tables 23-28, the application of Eco-CACC to a single dedicated lane on the SR-91 E Freeway led to significant savings at the highest tested volumes. Although additional testing needs to be conducted at different volumes, the initial results indicate that a single dedicated lane makes little to no difference in terms of travel time and energy at relatively low volumes. Just as with network 2 (i.e., IHFN), there was a substantial increase to overall capacity for network 3. When the vehicle spacing parameter was set to 5 meters, the benefits ranged from 0-42% for travel time and 0-19% for energy. Likewise, when the vehicle spacing parameter was set to 15 meters, the benefits ranged from 0-24% for travel time and 0-13% for energy. The results indicate that the primary difference between vehicle following distances of 5 and

15 meters is the resulting increase in capacity. The results at the highest volume tested, (37,000 vehicles), indicate that the system was within its capacity for the 5 meter test, but beyond its capacity for the 15 meter test. Consequently, additional testing must be conducted at higher volumes in order to quantify the difference in capacity between vehicle spacing of 5 and 15 meters.

It is important to note that the results shown do not represent the maximum achievable benefits possible for applying Eco-CACC in a single dedicated lane. Additional testing at higher traffic volumes is necessary in order to ascertain the new capacity for a given traffic network. The increase in capacity is related to the increase in traffic density caused by platooning. One remarkable feature of the aforementioned results is that the application of Eco-CACC to only a single lane still caused a substantial increase in network capacity. The application of Eco-CACC to more than one lane should theoretically result in even more dramatic increases in roadway capacity.

4.1.5.4.1.1 Part 1 Vehicle Spacing Parameter of 5 Meters

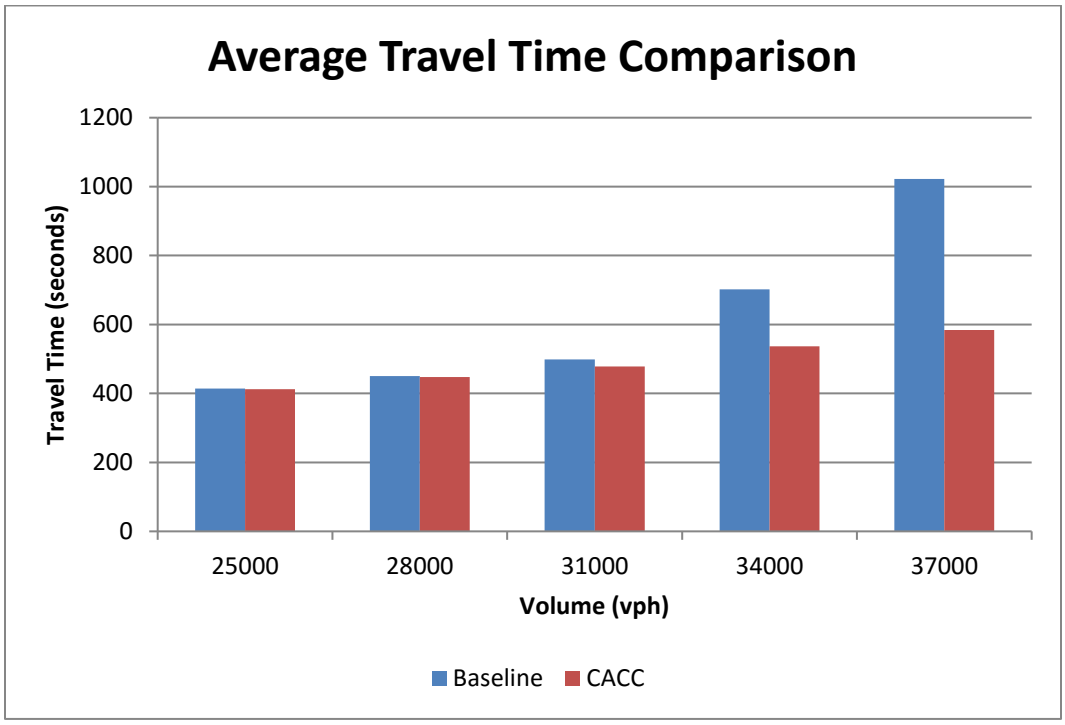

Figure 57: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates, 5m Vehicle Spacing

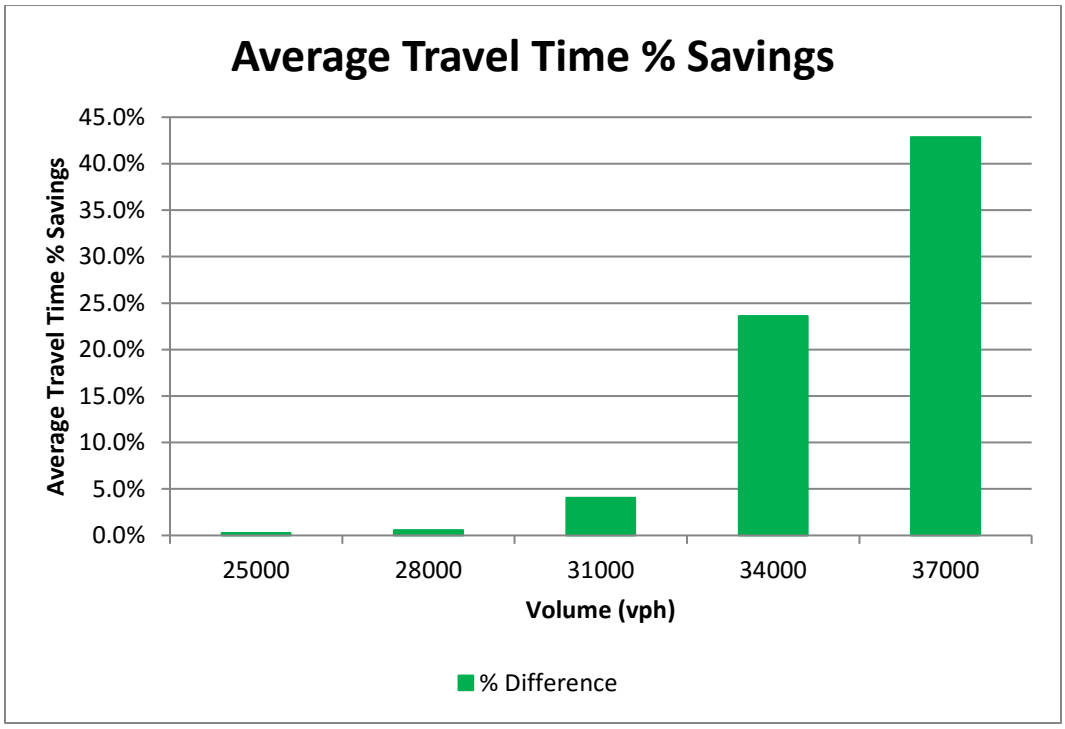

Figure 58: Average Travel Time % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 5m Vehicle Spacing

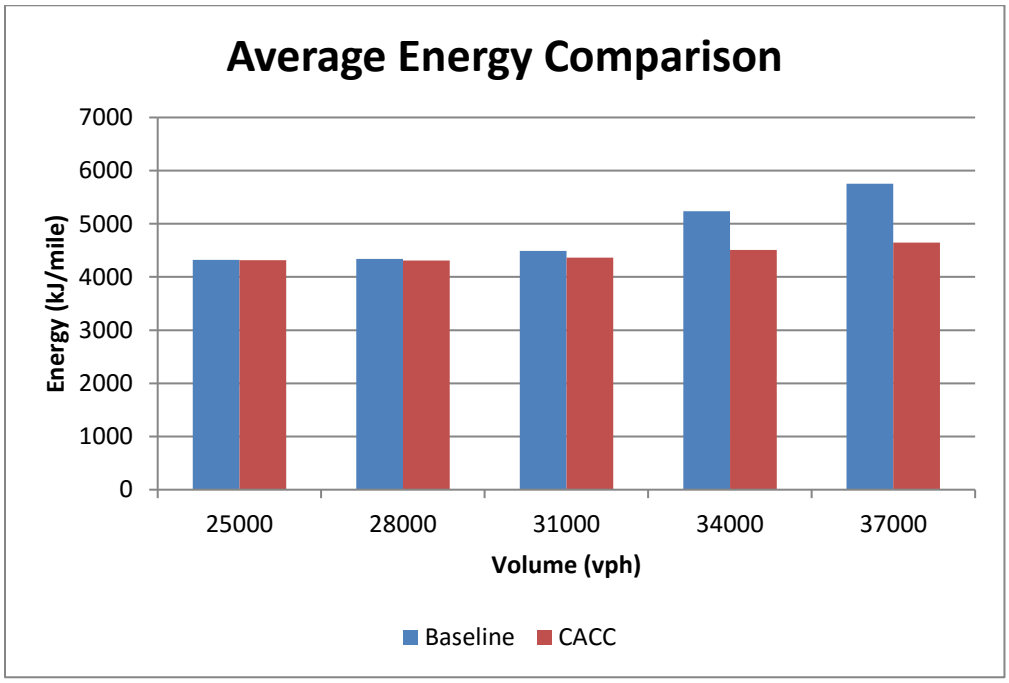

Figure 59: Energy vs. Traffic Volume for 0% & 100% Penetration Rates, 5m Vehicle Spacing

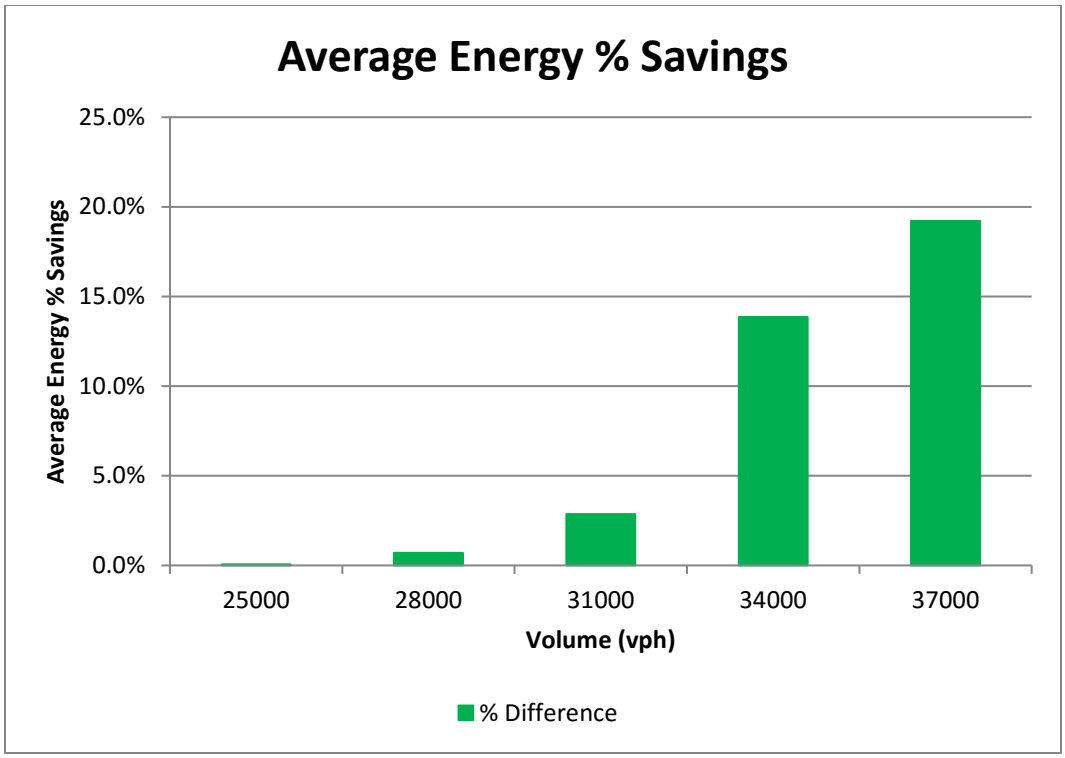

Figure 60: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 5m Vehicle Spacing

Table 23: Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	*Energy (kJ/mi)*	*CO2 (g/mi)*	*CO (g/mi)*	*HC (g/mi)*	*NOx (g/mi)*	*PM2.5 (g/mi)*	*VHT (s/veh)*
25000	4320.6885	310.5274	3.8453	0.1404	0.7533	0.0143	413.8942
28000	4341.5825	312.0290	3.9371	0.1421	0.7579	0.0145	450.1636
31000	4491.1921	322.7814	4.4314	0.1538	0.7859	0.0166	498.2444
34000	5235.4219	376.2688	5.9993	0.2046	0.8833	0.0245	702.3228
37000	5754.3728	413.5655	6.5284	0.2355	0.9220	0.0274	1022.4019

Table 24: Eco-CACC, Traffic Volume Sensitivity Analysis Results, 5m Vehicle Spacing

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
25000	4317.8101	310.3205	3.8630	0.1407	0.7530	0.0144	412.7937
28000	4310.7868	309.8157	3.9166	0.1411	0.7535	0.0143	447.6072
31000	4362.7826	313.5526	4.1373	0.1456	0.7653	0.0150	478.1213
34000	4509.7201	324.1130	4.5656	0.1566	0.7885	0.0166	536.5902
37000	4648.5840	334.0931	5.0162	0.1670	0.8161	0.0184	584.4299

Table 25: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results, 5m Vehicle Spacing

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
25000	0.07%	0.07%	-0.46%	-0.20%	0.04%	-0.45%	0.27%
28000	0.71%	0.71%	0.52%	0.72%	0.59%	1.56%	0.57%
31000	2.86%	2.86%	6.64%	5.32%	2.61%	9.21%	4.04%
34000	13.86%	13.86%	23.90%	23.43%	10.72%	32.32%	23.60%
37000	19.22%	19.22%	23.16%	29.07%	11.49%	33.04%	42.84%

4.1.5.4.1.2 Part 2 Vehicle Spacing Parameter of 15 Meters

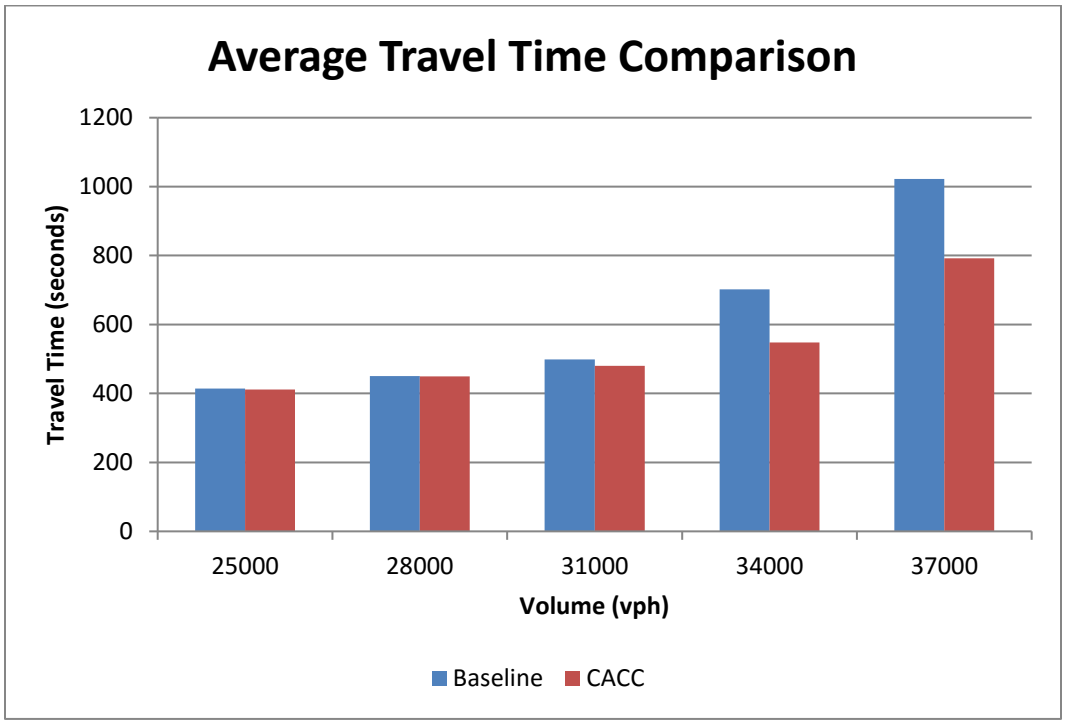

Figure 61: Average Travel Time vs. Traffic Volume for 0% & 100% Penetration Rates, 15m Vehicle Spacing

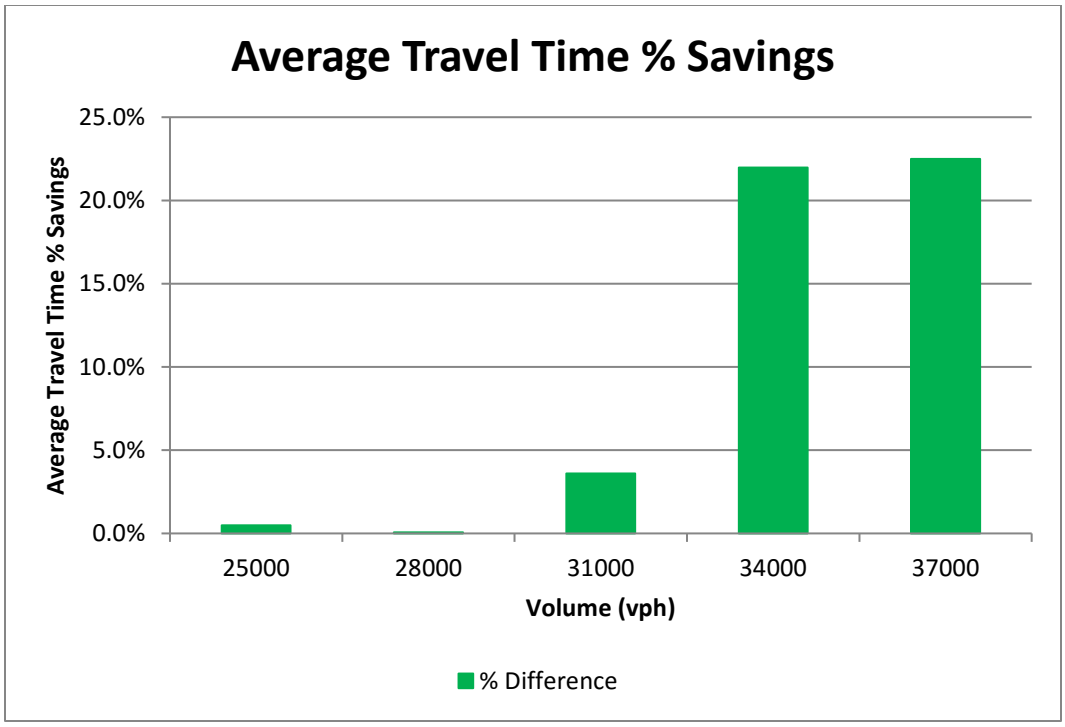

Figure 62: Average Travel Time % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 15m Vehicle Spacing

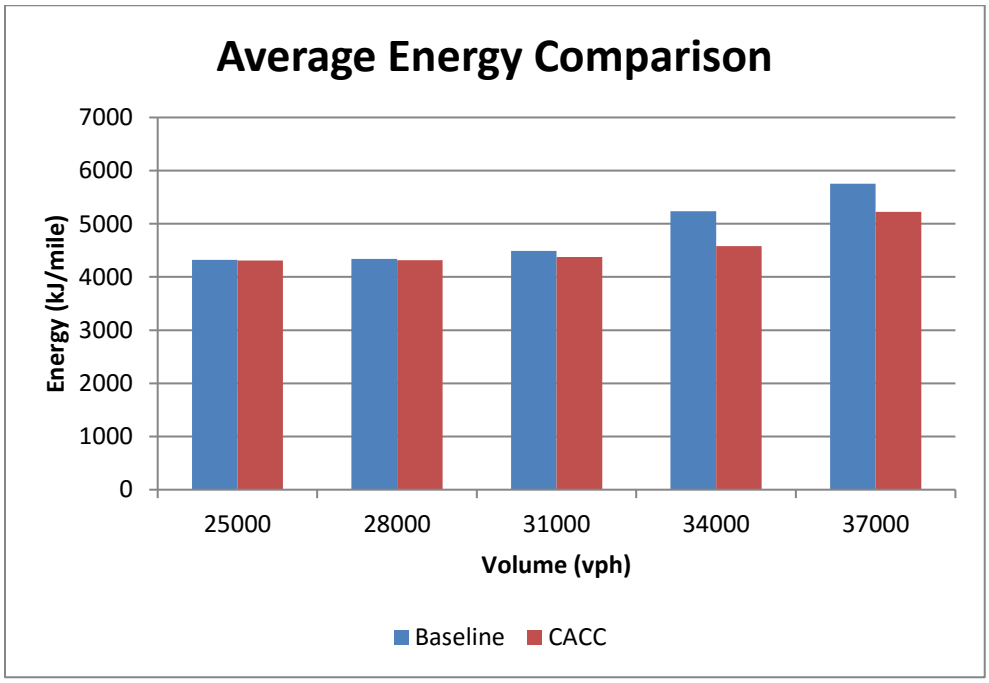

Figure 63: Energy vs. Traffic Volume for 0% & 100% Penetration Rates, 15m Vehicle Spacing

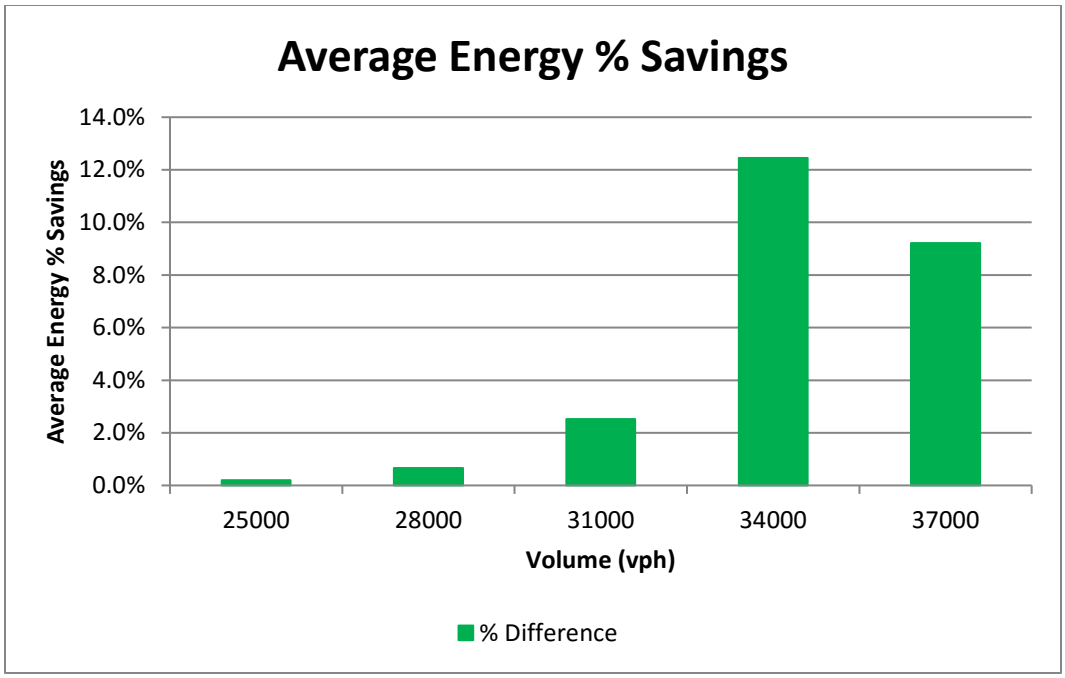

Figure 64: Energy % Savings vs. Traffic Volume for 100% vs. 0% Penetration Rates, 15m Vehicle Spacing

Table 26: Baseline, Traffic Volume Sensitivity Analysis Results

Volume (vph)	*Energy (kJ/mi)*	*CO2 (g/mi)*	*CO (g/mi)*	*HC (g/mi)*	*NOx (g/mi)*	*PM2.5 (g/mi)*	*VHT (s/veh)*
25000	4320.6885	310.5274	3.8453	0.1404	0.7533	0.0143	413.8942
28000	4341.5825	312.0290	3.9371	0.1421	0.7579	0.0145	450.1636
31000	4491.1921	322.7814	4.4314	0.1538	0.7859	0.0166	498.2444
34000	5235.4219	376.2688	5.9993	0.2046	0.8833	0.0245	702.3228
37000	5754.3728	413.5655	6.5284	0.2355	0.9220	0.0274	1022.4019

Table 27: Eco-CACC, Traffic Volume Sensitivity Analysis Results, 15m Vehicle Spacing

Volume (vph)	*Energy (kJ/mi)*	*CO2 (g/mi)*	*CO (g/mi)*	*HC (g/mi)*	*NOx (g/mi)*	*PM2.5 (g/mi)*	*VHT (s/veh)*
25000	4312.2076	309.9178	3.8214	0.1400	0.7512	0.0142	411.8914
28000	4313.1120	309.9828	3.8400	0.1401	0.7529	0.0141	449.8930
31000	4378.0505	314.6500	4.0398	0.1446	0.7664	0.0150	480.3299
34000	4583.5199	329.4169	4.6079	0.1595	0.7997	0.0172	547.9909
37000	5224.1798	375.4609	5.7884	0.2000	0.8899	0.0245	792.3982

Table 28: % Improvement of Eco-CACC over Baseline, Traffic Volume Sensitivity Analysis Results, 15m Vehicle Spacing

Volume (vph)	Energy (kJ/mi)	CO2 (g/mi)	CO (g/mi)	HC (g/mi)	NOx (g/mi)	PM2.5 (g/mi)	VHT (s/veh)
25000	0.20%	0.20%	0.62%	0.26%	0.28%	0.69%	0.48%
28000	0.66%	0.66%	2.47%	1.44%	0.66%	2.43%	0.06%
31000	2.52%	2.52%	8.84%	5.96%	2.48%	9.64%	3.60%
34000	12.45%	12.45%	23.19%	22.01%	9.46%	29.93%	21.97%
37000	9.21%	9.21%	11.33%	15.08%	3.48%	10.54%	22.50%

4.1.5.4.2 Traffic Network Savings vs. Dedicated Lane Savings

The average energy savings considering the entire network differs from the average energy savings experienced in the dedicated lane. Energy data for the dedicated lane was defined as the aggregation of energy data for vehicles while they were on the dedicated lane. Therefore, if a vehicle changes lanes into the dedicated lane, energy data is accumulated as long as the vehicle remains in the dedicated lane. If a vehicle changes lanes from the dedicated lane, then that vehicle no longer contributes energy data to the dedicated lane analysis. Note that vehicles with different routes use the dedicated lane. Furthermore, based on the aforementioned definition, there is no corresponding dedicated lane travel time definition due to lane changes and diverse routes. In addition to showing the dedicated lane savings, the average savings for the remaining lanes are also shown in Figures 65 and 67. Figures 66 and 68 show the relative benefit of choosing the dedicated lane over a non-dedicated lane.

The results also demonstrate that for the tested traffic volumes, the dedicated lane is the most energy efficient lane. In fact, the dedicated lane was 3-12% more energy efficient than the non-dedicated lanes, and 3-26% more energy efficient relative to the average baseline scenario lane. In addition to the direct benefits that Eco-CACC brings to participating vehicles, other vehicles and lanes indirectly benefited. The non-dedicated lanes saw a reduction in energy consumption ranging from 0-16%.

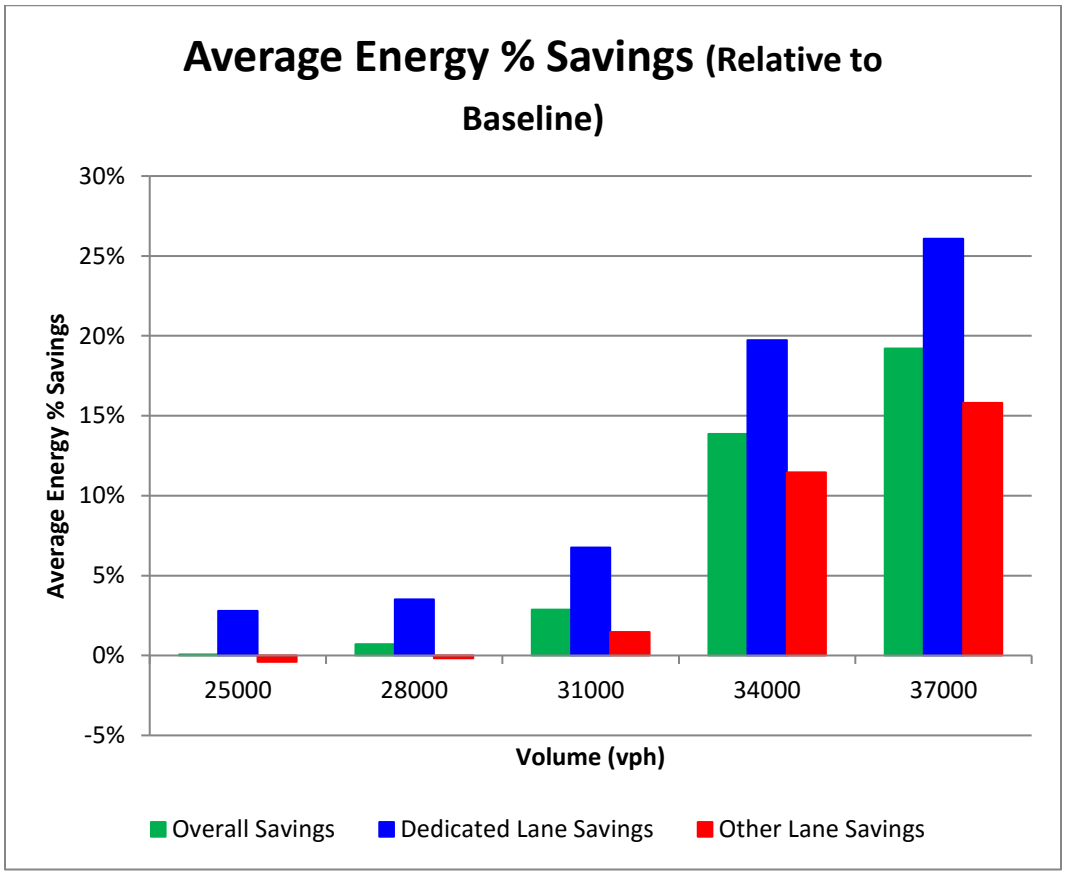

Figure 65: Dedicated Lane Energy Savings vs. Traffic Volume, 5m Vehicle Spacing

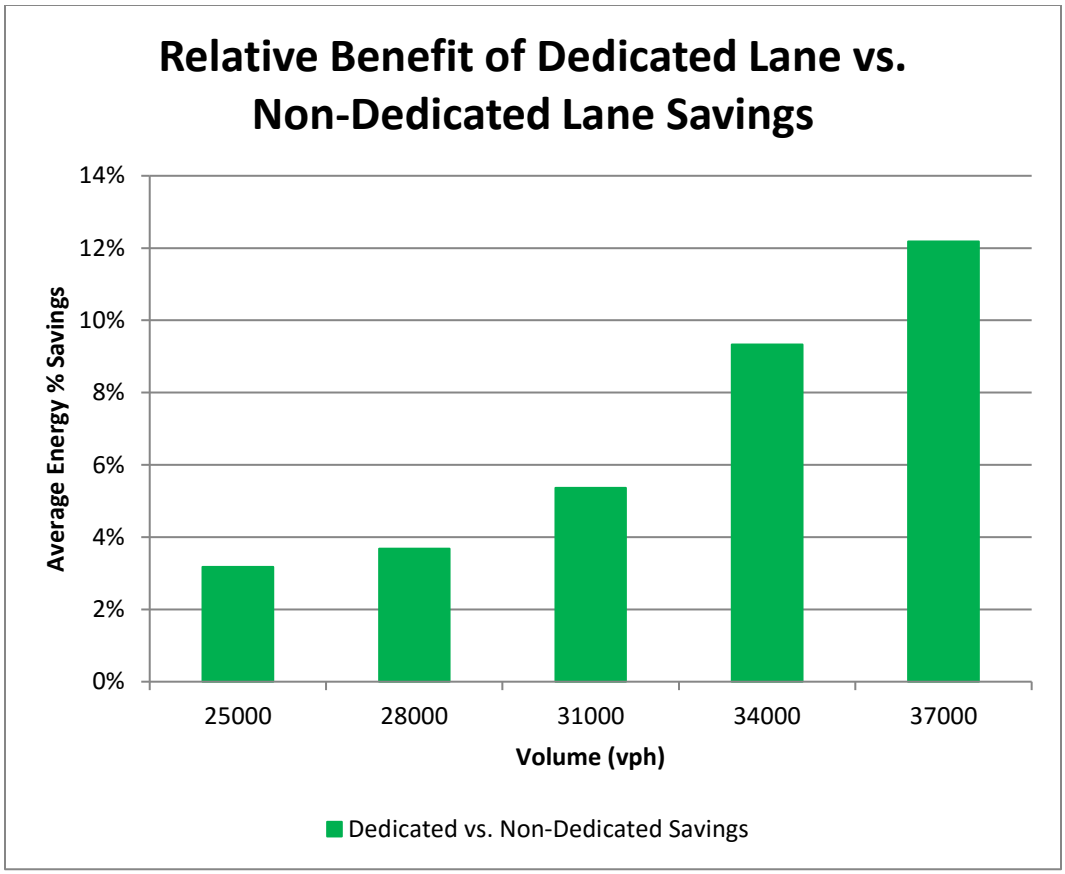

Figure 66: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings, 5m Vehicle Spacing

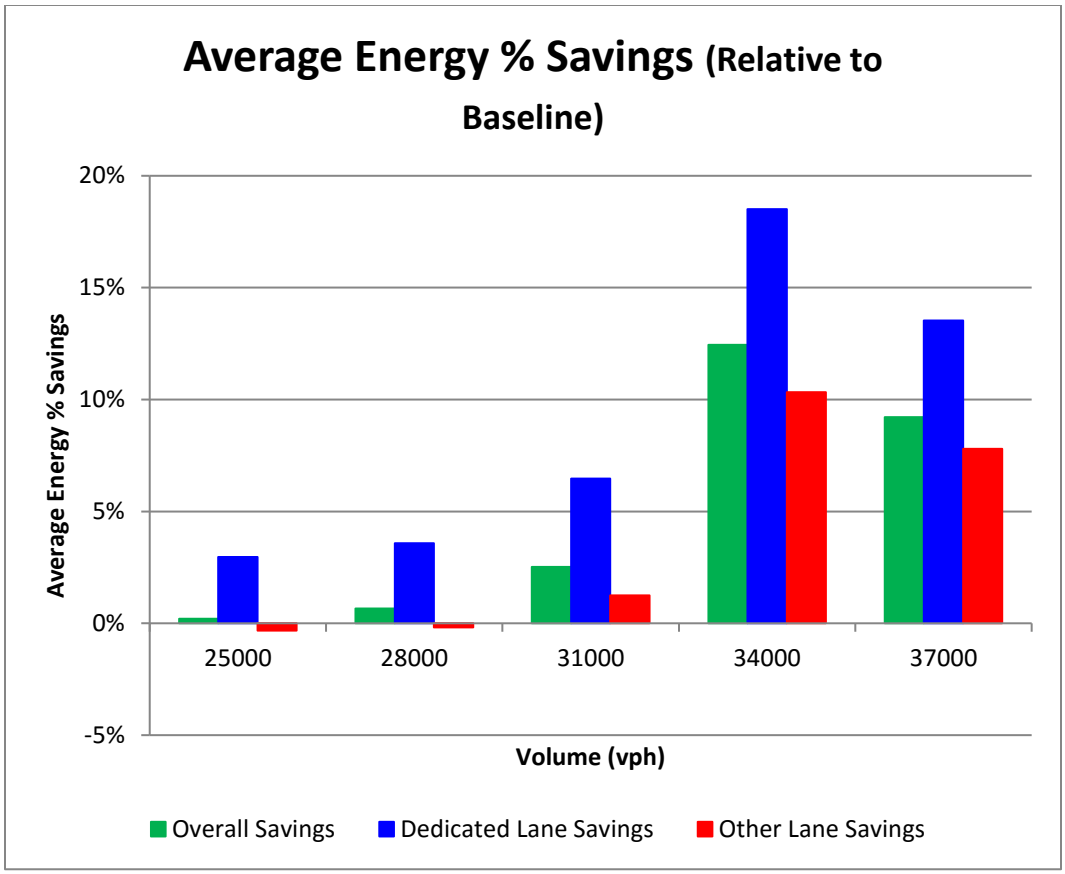

Figure 67: Dedicated Lane Energy Savings vs. Traffic Volume, 15m Vehicle Spacing

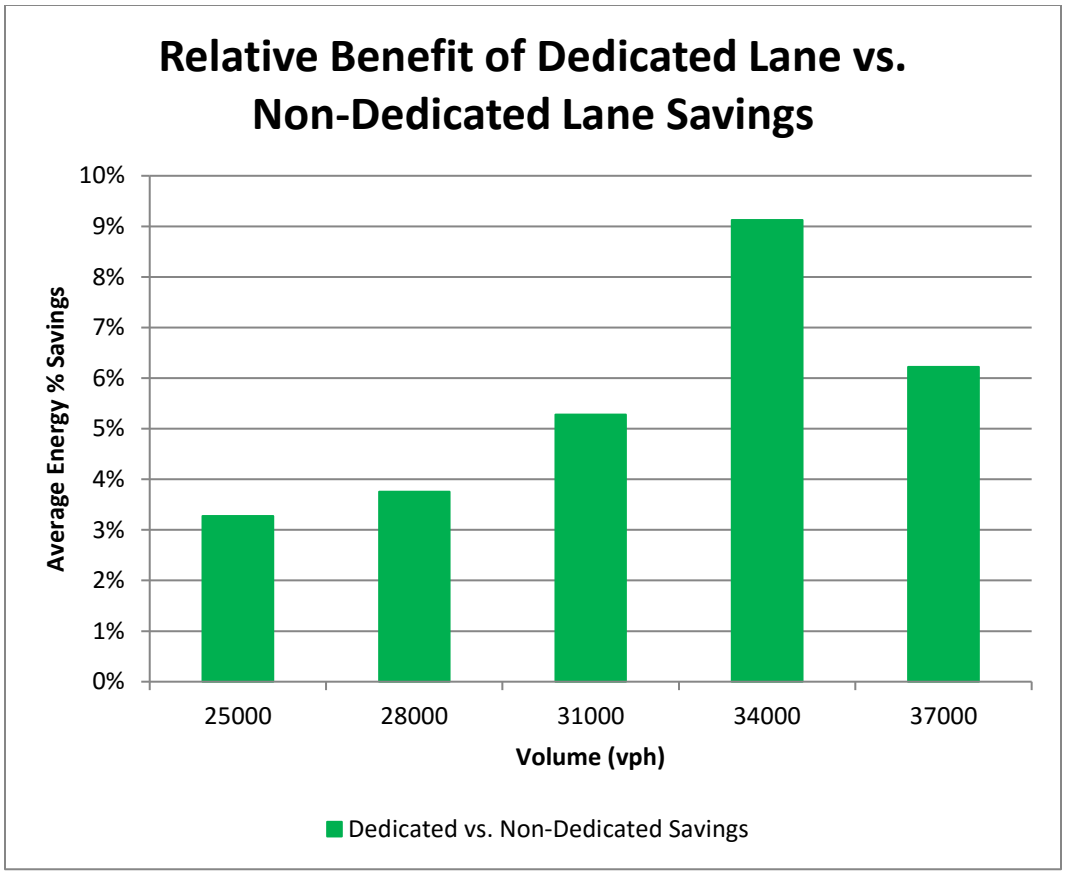

Figure 68: Relative Benefit of Dedicated Lane vs. Non-Dedicated Lane Savings, 15m Vehicle Spacing

4.1.5.4.3 Benefits for Mainstream Vehicles Choosing the Dedicated Lane

In addition to aggregating data by lane, data may also be aggregated for vehicles following a specific route. The selected route which made the most use of the dedicated lane was the mainstream route. The mainstream route was defined as the only two routes not including on or off ramps. Along the mainstream route, data collection was further divided among vehicles which chose to use the dedicated lane, and vehicles which did

not choose the dedicated lane. The relative travel time and energy benefits of choosing the dedicated lane versus a non-dedicated lane are shown in Figures 69 and 70.

Equipped vehicles which chose to use the dedicated lane obtained a 10-19% benefit in reduced travel time, with a -1-5% benefits in energy savings.

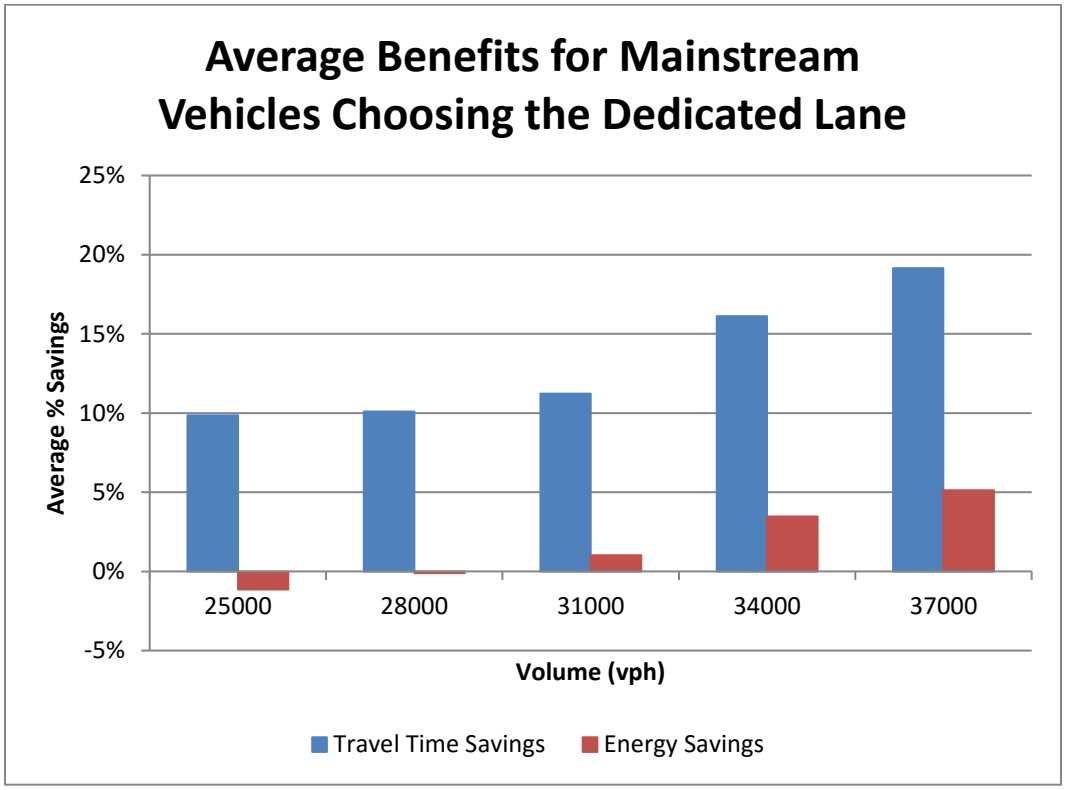

Figure 69: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane, 5m Vehicle Spacing

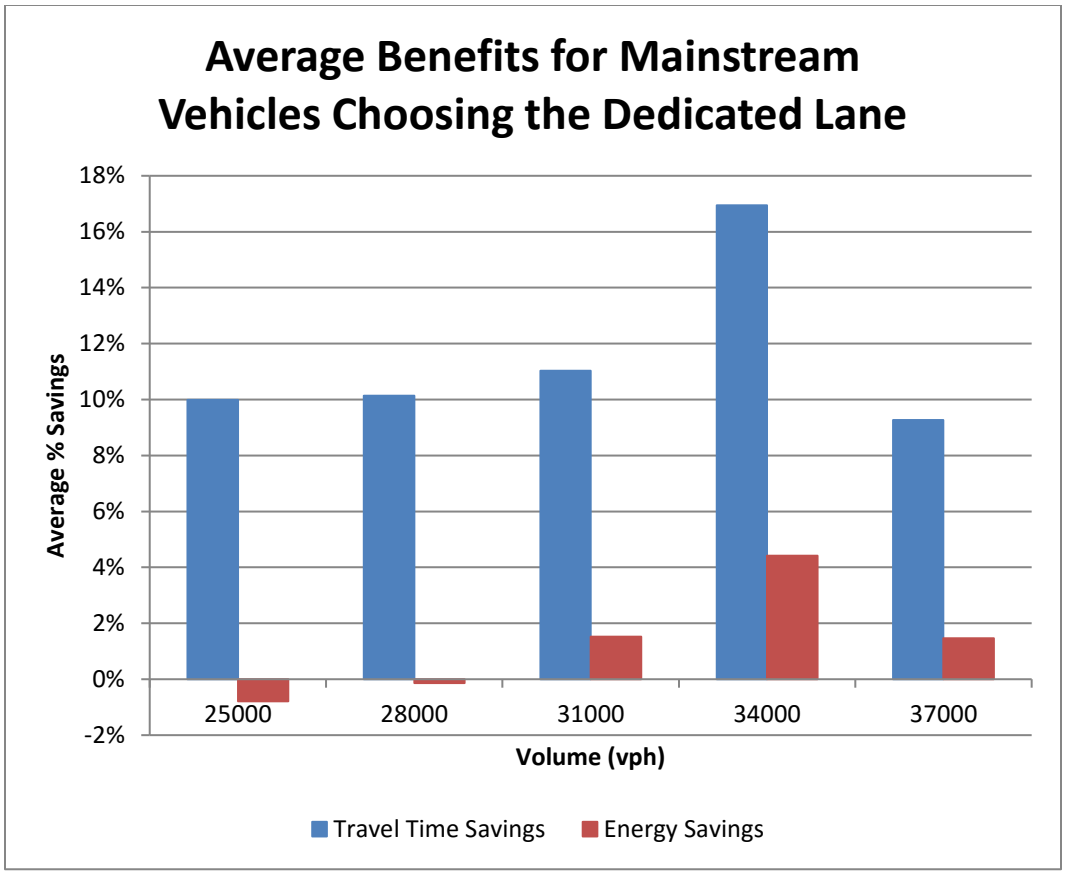

Figure 70: Average Benefits for Mainstream Vehicles Choosing the Dedicated Lane, 15m Vehicle Spacing

4.2 The Role of Acceleration and Deceleration in Eco-tuning of Eco-Cooperative Adaptive Cruise Control

Prior sections examined the role of intra-platoon spacing and triggering distance in effecting the eco-benefits of Eco-CACC. Additional parameters which may affect the performance of Eco-CACC include the acceleration and deceleration constraints placed on the gap-closing controller. Generally, the acceleration and deceleration constraints are chosen based on driver comfort, safety, and controller stability. In the case of Eco-

CACC, the selection of the values for the acceleration and deceleration constraints is also influenced by the potential environmental benefits of choosing particular values. The following section will present the results of an experiment designed to test the role of acceleration and deceleration constraints in the tuning of an Eco-CACC controller.

An acceleration test and a deceleration test were conducted on the SR-91 EB Freeway network in Paramics. The traffic volume was set to approximately 31,000 vehicles released over 2 hours. The triggering distance was set at 40 meters with the intra-platoon spacing set to 15 meters. In addition, the penetration rate of Eco-CACC technology was set to 100%. For the acceleration test, the deceleration constraint was set at -7.5 m/s^2, and the set of acceleration constraint values tested was 2, 3.5, and 5 m/s^2. For the deceleration test, the acceleration constraint was set at 3.5 m/s^2, and the set of deceleration constraint values tested was -9, -7.5, and -6 m/s^2.

The results for the acceleration test are shown in Figures 71-72 and Tables 29-30 below. Likewise, the results for the deceleration test are shown in Figures 73-74 and Tables 31-32 below. From left to right, the first set of columns in Figure 71 depict the average travel time of all vehicles in the traffic network. The second set of columns shows the average travel time for vehicles which used the dedicated lane for any portion of their trip in the traffic network. The third set of columns shows the average travel time for vehicles which did not use the dedicated lane for any portion of their trip in the traffic network. The fourth set of columns shows the average travel time for vehicles passing through the entire length of the freeway network while adopting to use the dedicated lane.

The fifth set of columns shows the average travel time for vehicles passing through the entire length of the freeway network without using the dedicated lane. A similar description may be applied to Figure 72 in terms of energy. The results appear to indicate that neither travel time nor energy is substantially impacted by the selection of different acceleration or deceleration constraints. The difference in values for different acceleration and deceleration constraints is less than 1 percent. Although a lower acceleration rate minimizes the energy rate, the gap closing maneuver takes longer than with a higher acceleration rate. Consequently, the overall average energy rate remains roughly the same for different values of acceleration and deceleration constraints.

In this chapter, a series of experiments were conducted in order to explore how environmental benefits are affected with changes in controller settings and traffic conditions. Various values of the triggering distance, the intra-platoon spacing, the acceleration constraint, and the deceleration constraint do not significantly impact environmental benefits. In contrast, energy benefits are maximized when the penetration rate of Eco-CACC technology is greater than or equal to 60% and when the traffic volume is relatively high. Due to the increase in freeway capacity, the use of a small intra-platoon spacing value coupled with a high traffic volume also leads to substantial energy savings.

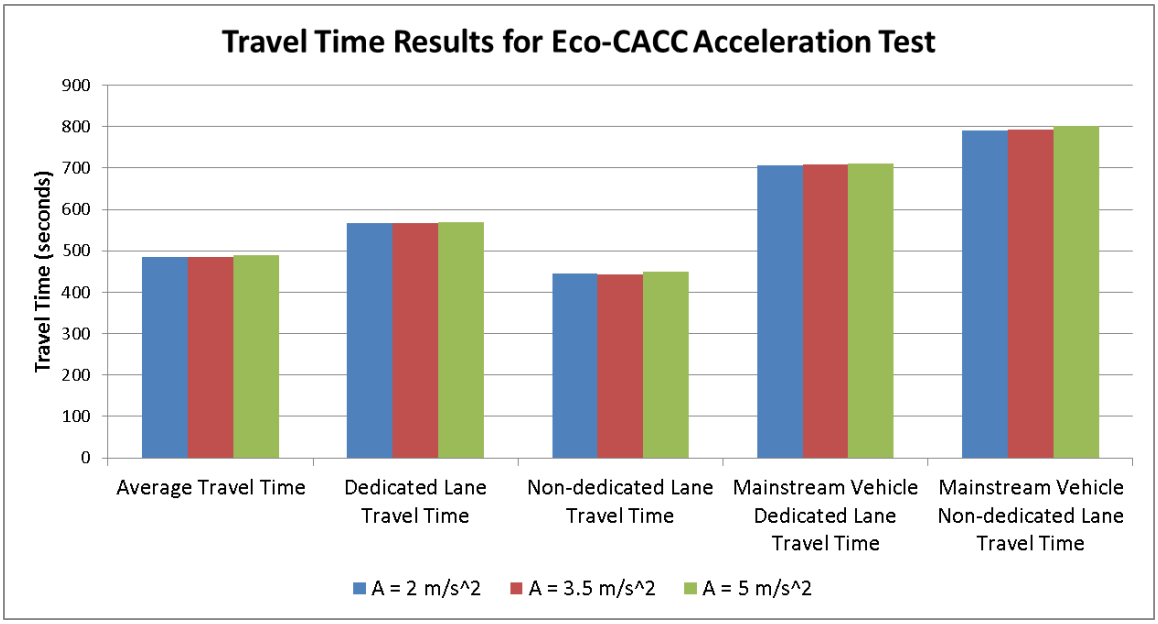

Figure 71: Travel Time Results for Eco-CACC Acceleration Test

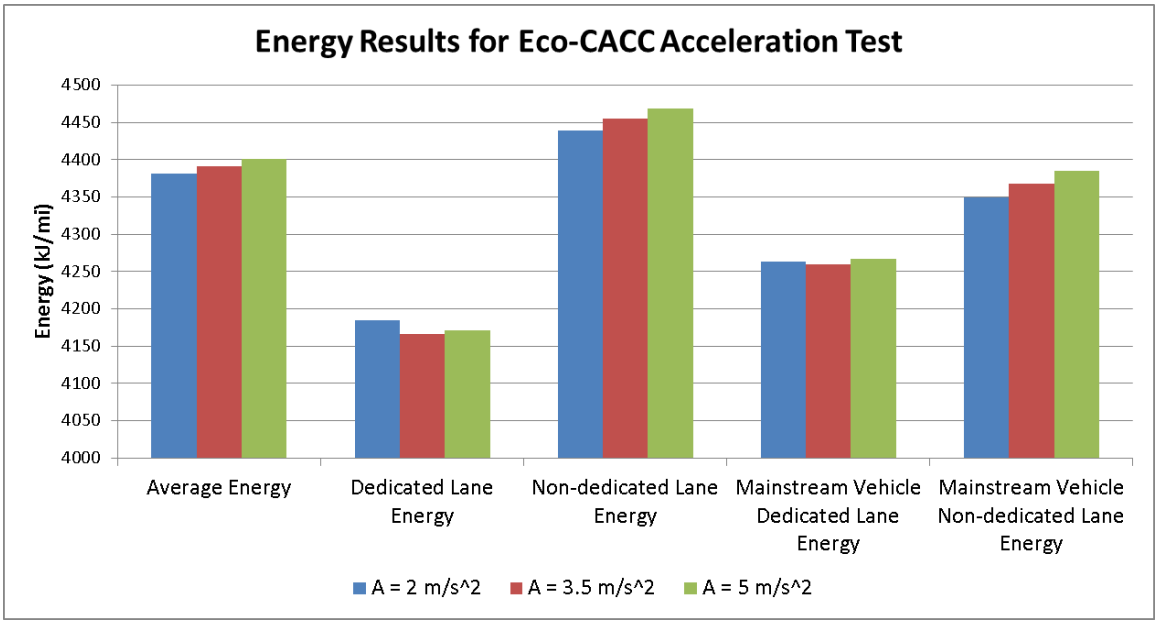

Figure 72: Energy Results for Eco-CACC Acceleration Test

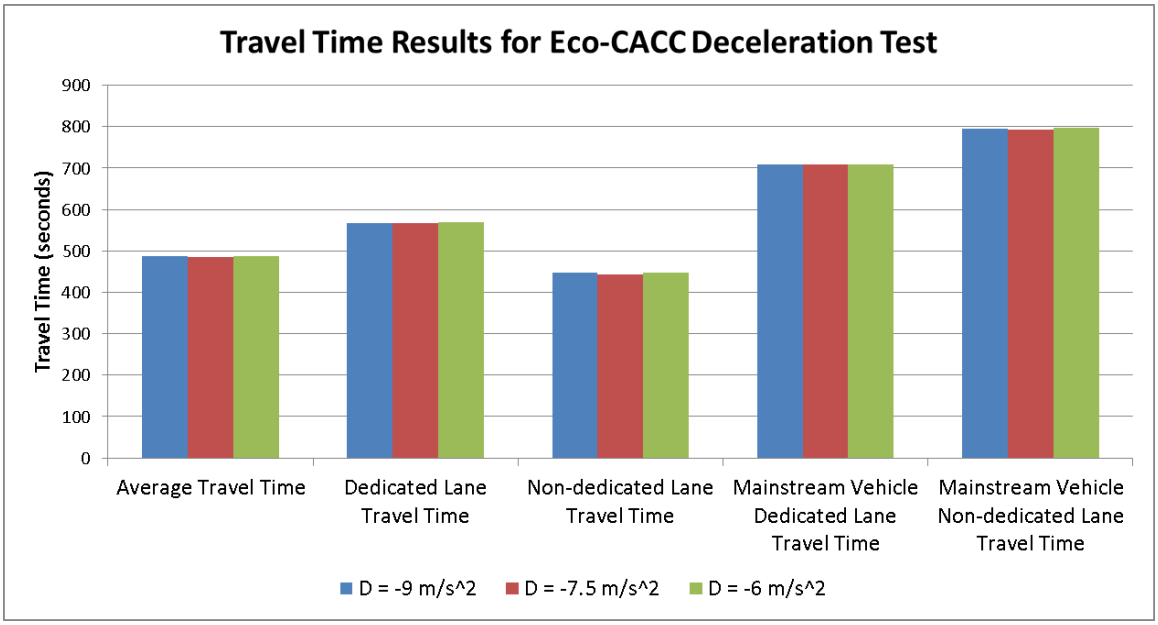

Figure 73: Travel Time Results for Eco-CACC Deceleration Test

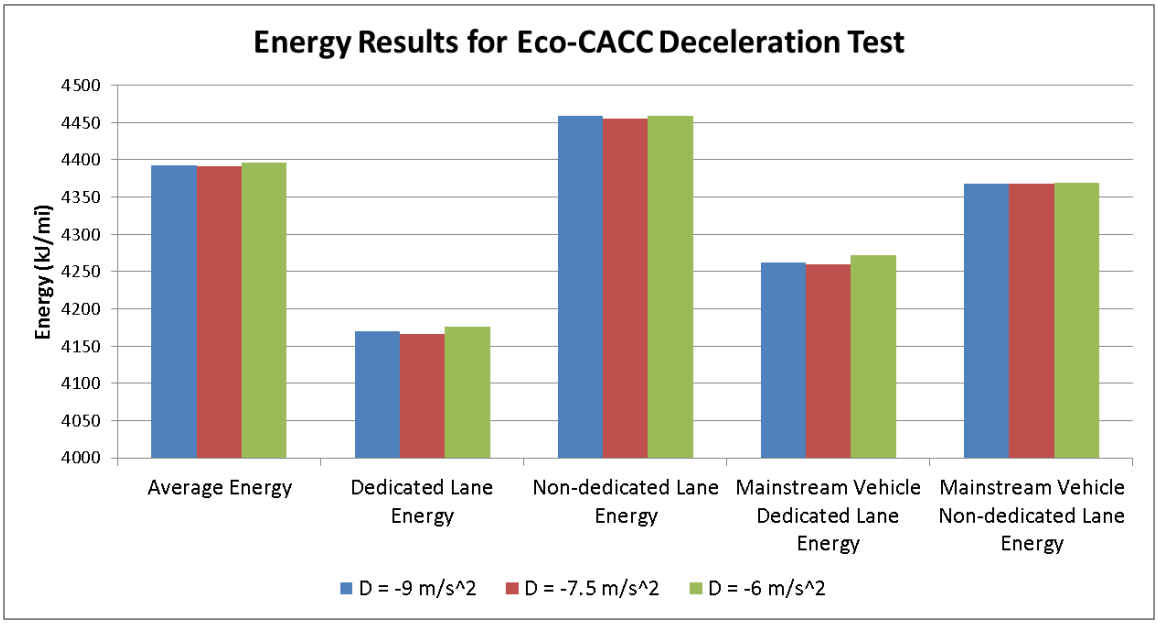

Figure 74: Energy Results for Eco-CACC Deceleration Test

Table 29: Travel Time (seconds) Results for Eco-CACC Acceleration Test

Acc-eleration (m/s^2)	Average Travel Time	Dedicated Lane Travel Time	Non-dedicated Lane Travel Time	Mainstream Vehicle Dedicated Lane Travel Time	Mainstream Vehicle Non-dedicated Lane Travel Time
2	484.5572	566.3803	445.2466	706.8880	790.1182
3.5	484.0903	568.1923	443.6583	708.7223	793.3030
5	489.2967	569.3926	450.3007	710.2297	802.2240

Table 30: Energy (kJ/mi) Results for Eco-CACC Acceleration Test

Acc-eleration (m/s^2)	Average Energy	Dedicated Lane Energy	Non-dedicated Lane Energy	Mainstream Vehicle Dedicated Lane Energy	Mainstream Vehicle Non-dedicated Lane Energy
2	4381.8340	4184.2952	4439.6064	4263.8590	4349.3294
3.5	4391.0021	4166.6851	4455.5983	4259.2094	4367.6933
5	4401.6143	4170.5899	4469.0691	4267.3856	4385.2374

Table 31: Travel Time (seconds) Results for Eco-CACC Deceleration Test

Dec-eleration (m/s^2)	Average Travel Time	Dedicated Lane Travel Time	Non-dedicated Lane Travel Time	Mainstream Vehicle Dedicated Lane Travel Time	Mainstream Vehicle Non-dedicated Lane Travel Time
-9	486.6348	568.1720	447.3896	709.8134	794.5643
-7.5	484.0903	568.1923	443.6583	708.7223	793.3030
-6	486.5168	568.6717	447.0376	708.0845	797.1637

Table 32: Energy (kJ/mi) Results for Eco-CACC Deceleration Test

Dec-eleration (m/s^2)	Average Energy	Dedicated Lane Energy	Non-dedicated Lane Energy	Mainstream Vehicle Dedicated Lane Energy	Mainstream Vehicle Non-dedicated Lane Energy
-9	4392.7206	4169.7690	4458.3919	4262.6707	4367.4481
-7.5	4391.0021	4166.6851	4455.5983	4259.2094	4367.6933
-6	4395.8224	4175.8331	4459.5897	4271.8745	4369.6625

5. Integrating Infrastructure Enhancement Techniques with Connected Vehicles

5.1 Adaptive/Optimal Signal Control with Intelligent Vehicles

One of the emerging techniques for enhancing infrastructure is to equip infrastructure with the hardware necessary for receiving wireless information broadcasted by Connected Vehicles (CVs). In the context of arterial traffic intersections, the intersection can receive real-time state information from nearby CVs and dynamically adjust the signal timing to maximize the efficiency of the intersection. An adaptive signal control system for use with CVs is presented in the sections below. The system is applied in the context of an isolated intersection as well as for a corridor of traffic intersections.

5.1.1 Isolated Intersection Connected Vehicle Signal Optimization

5.1.1.1 Application Description

Conventional adaptive signal control strategies make use of point detection sensors such as inductive loop detectors (ILDs), video sensors, or radar/LiDAR sensors to adjust the signal timing based on the limited available knowledge of incoming traffic. Each of the aforementioned sensors has one or more significant drawbacks such as accuracy, occlusion, and degraded performance due to adverse lighting or weather conditions. Each of the traditional sensor drawbacks is circumvented with the use of wireless CV technology. The following section describes an adaptive signal control

optimization strategy based on using CV technology to build a complete and accurate picture of real-time traffic conditions near an arterial intersection.

5.1.1.1.1 System Introduction

Considering traffic at an intersection to be a multi-agent system (MAS), the signal phase and timing may be controlled to improve overall traffic efficiency. A multi-agent system is a computerized system composed of multiple intelligent agents interacting within an environment [Niazi and Hussain, 2011]. At a given intersection, two types of agents may be considered: 1) Vehicle Agents (VA), and an 2) Intersection Management Agent (IMA). The role of the VA is defined as including communicating ego information to the IMA. The role of the IMA is defined as including communicating with all VA's within a communication radius, and determining the optimal signal timing. The overall system architecture is presented in the following subsection. Next, an extension of the dual-ring traffic controller is introduced, followed by a description of the signal timing optimization method utilized.

5.1.1.1.2 System Architecture

As shown in Figure 75, the system consists of multiple vehicle agents interacting with a single intersection management agent. Each IMA is intended to control a single intersection. Due to the nature of adaptive signal control, the following strategy is readily extended to multiple intersections and corridors. Alternative implementations may also include VAs communicating with each other.

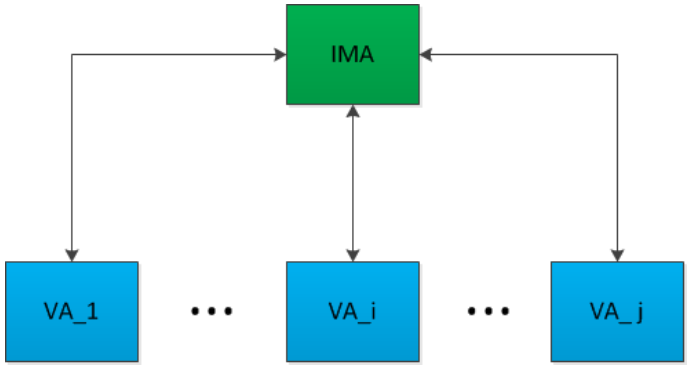

Figure 75: CV MAS Level 0 Diagram: Top-level System Architecture

The intersection management agent controls the traffic signal lights based on received information from all of the VAs within communication range of the intersection. As indicated in Figure 76, the IMA makes use of several signal timing constraints to determine when the signal timing needs to be changed. These constraints include the minimum green time, maximum green time, yellow time, as well as the "all-red" duration. If none of these constraints are in effect, then the IMA re-evaluates the traffic environment every one second in order to determine if a change in signal timing is necessary for optimizing the user-defined Measure of Effectiveness (MOE). Among others, MOEs may include queue length, idling time, energy consumption, or number of stops. Based on the selected MOE, a VA may need to predict certain information in order to provide the IMA with input. For example, if the selected MOE is travel delay, then VAs need to predict Time-Of-Arrival (TOA) based on proposed Signal Phase and Timing (SPaT) plans. The VA actions are detailed in Figure 77. If a VA is within communication range of the intersection, it receives the stop bar location from the IMA.

Using the stop bar location, if a VA is approaching the traffic signal, it then sends the necessary ego information to the IMA (based on the selected system-wide MOE).

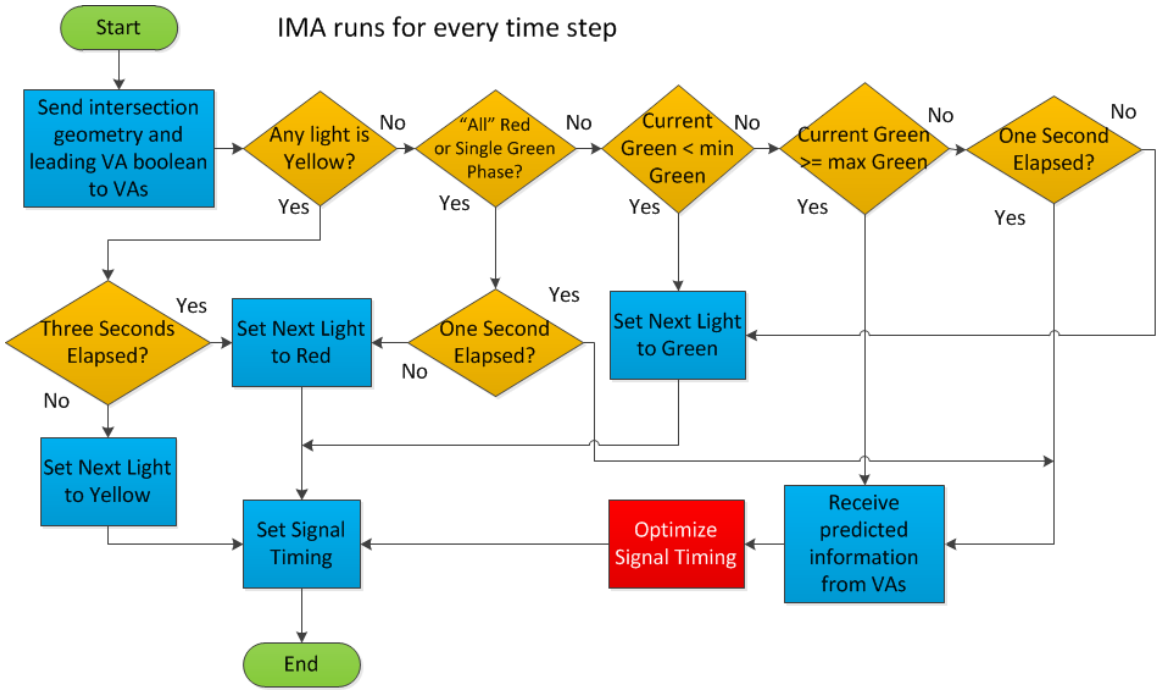

Figure 76: CV MAS Level 1 Diagram: Intersection Management Agent Flow Chart

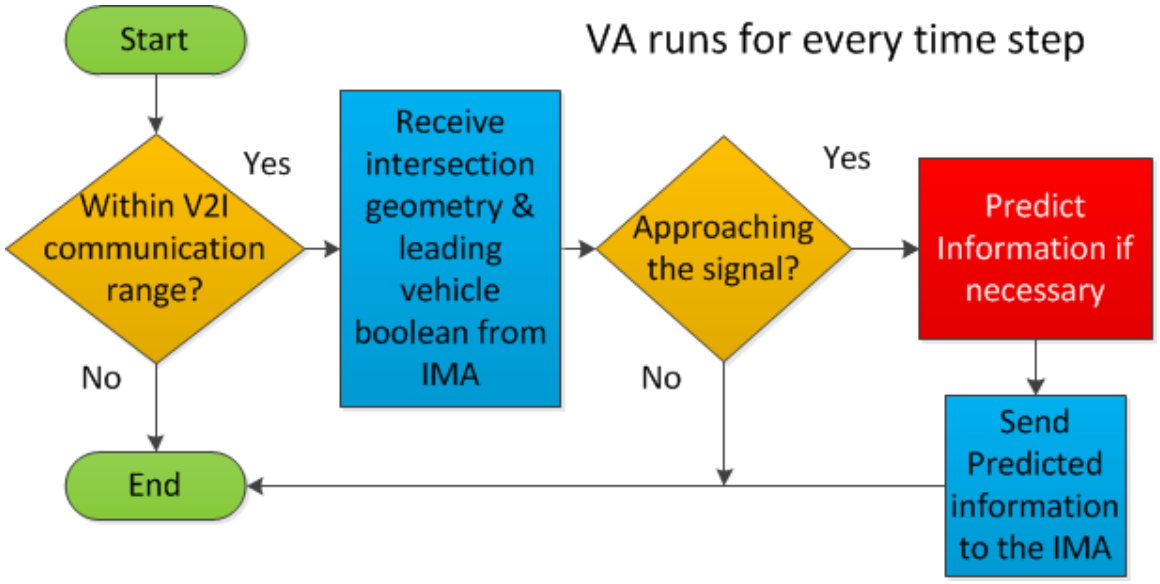

Figure 77: CV MAS Level 1 Diagram: Vehicle Agent Flow Chart

5.1.1.1.3 Fixed & Flexible Traffic Light State Machines

Perhaps the most common traffic controller used in the United States is the dual-ring National Electrical Manufacturing Association (NEMA) controller. Figure 78 includes the dual-ring controller and the corresponding NEMA signal phase diagram. The two "rings" correspond to two sets of self-conflicting phases, phases {1, 2, 3, 4} belonging to "Ring 1" and phases {5, 6, 7, 8} belonging to "Ring 2." At any given time instant, two signal phases are active, one from each ring. The two rings operate independently, with the restriction that the selected phases must be on the same side of the barrier (e.g. phases 2 and 7 cannot be active simultaneously). Main street phases are normally numbered as {1, 2, 5, 6}, while side street phases are typically numbered as {3, 4, 7, 8}. A typical background cycle consists of a fixed pairing and sequence of phases, with the main street movements being served prior to the side street movements. For a standard 4-leg intersection, there are four green phases per cycle, separated by appropriate yellow and red phases. Fixed signal timing uses pre-determined durations (splits) for each of the four green phases and uses the fixed sequence and combination of phases prescribed in Figure 78. As shown in Figure 78, signal operation starts with phases 1 and 5, followed by 2 and 6, 3 and 7, and 4 and 8, before repeating.

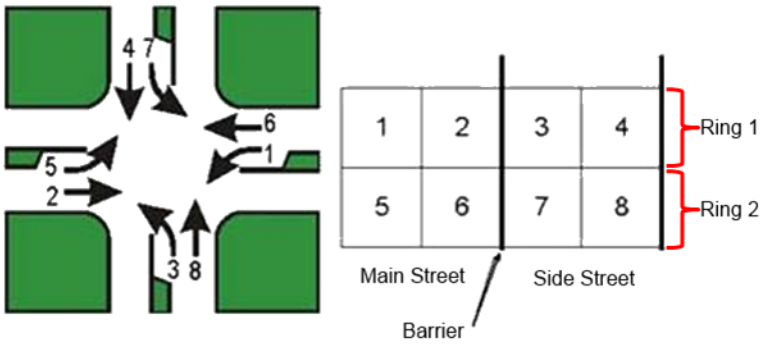

Figure 78: Signal Phase Diagram & Dual-Ring Controller, adapted from ["Signal Timing on a Shoestring", 2005]

To further illustrate the limitations of the fixed signal timing interpretation of the dual-ring controller, the fixed sequence of traffic signals may be represented using a finite state machine, as shown in Figure 79. Including yellow and red phases, there are a total of 9 unique states, with the "All" red phase repeated in the transition between every phase. Previous work was based on using the fixed sequence of traffic signals as prescribed by the dual-ring controller, and focused on optimizing the duration of each of the green splits [Kari, 2014]. However, a fully adaptive signal control paradigm should also consider optimizing phase sequence in addition to phase duration. Moreover, it is not necessary to have a strict coupling of phases such as 1 and 5, and 2 and 6. In fact, phase 1 may operate with either phase 5 or 6. By permitting the rings to operate independently, the dual-ring controller may be represented with a more advanced and flexible finite state machine, as shown in Figure 80. The red cylinder, labeled as the "All Red" state, represents the barrier, as well as the only link, between the main street and side street phases. There are four "green" states on each side of the barrier, for a total of eight "green" states. The main street half of the diagram in Figure 80 is shown in Figure

81. The side street half of the diagram in Figure 80 is nearly identical to the main street half, and is shown in Figure 82. "Green" colored states occur where two green phases are active. "Yellow" colored states occur where at least one signal phase is yellow. Finally, "Red" colored states occur where all traffic lights are red, or if all but one signal phase is red.

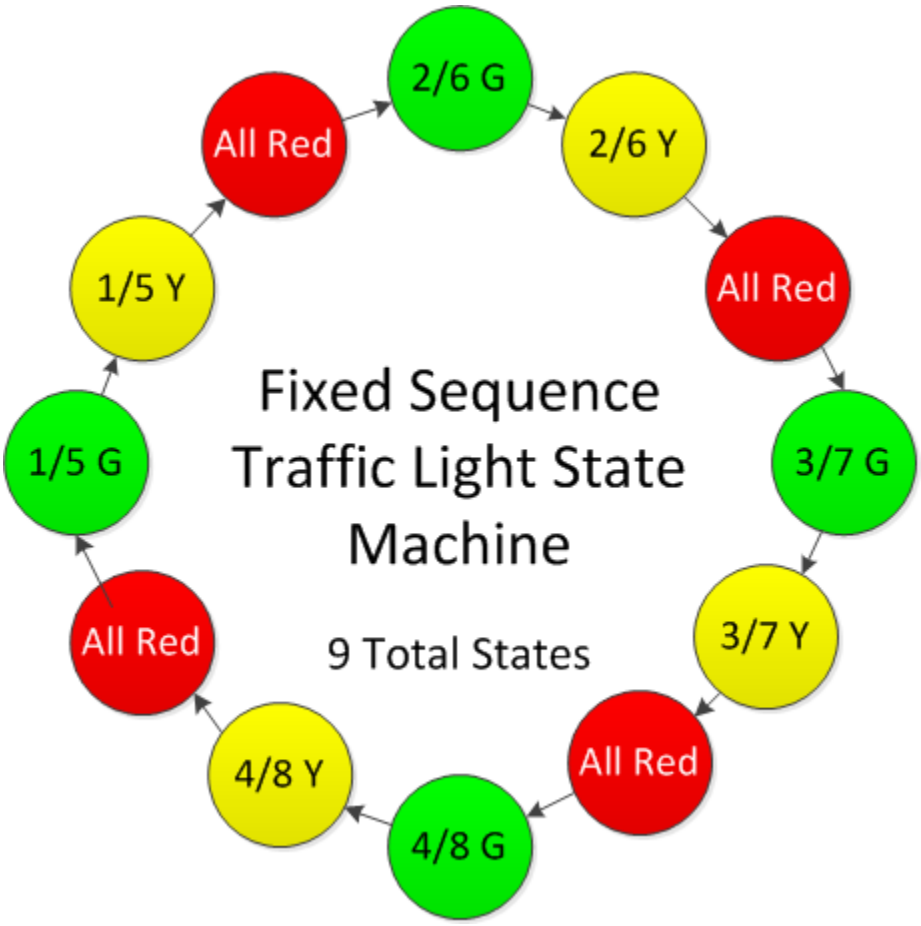

Figure 79: Fixed Sequence & Coupled Phase Dual-Ring Controller, Finite State Machine Representation

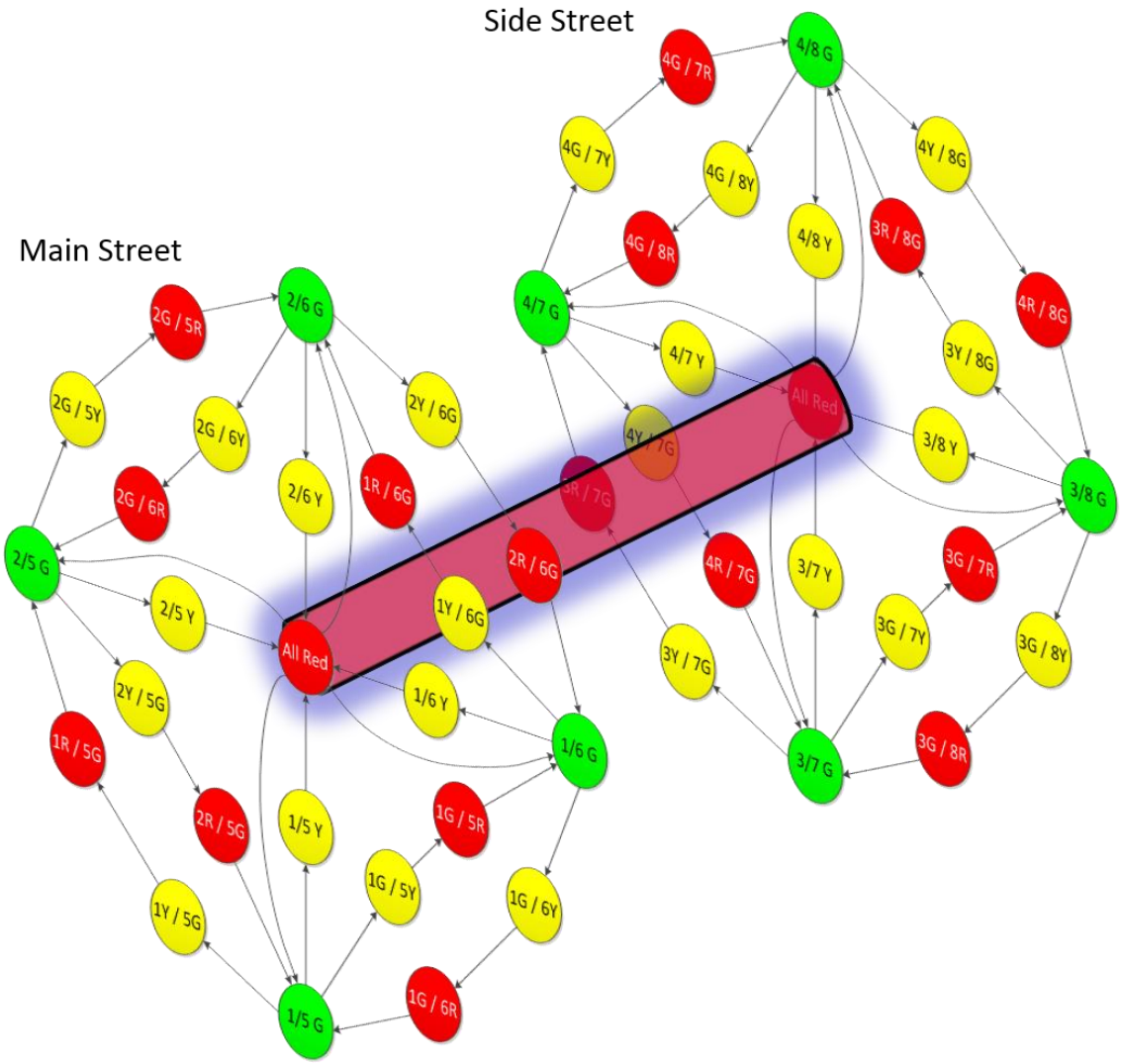

Figure 80: Flexible Dual-Ring Controller, Finite State Machine Representation

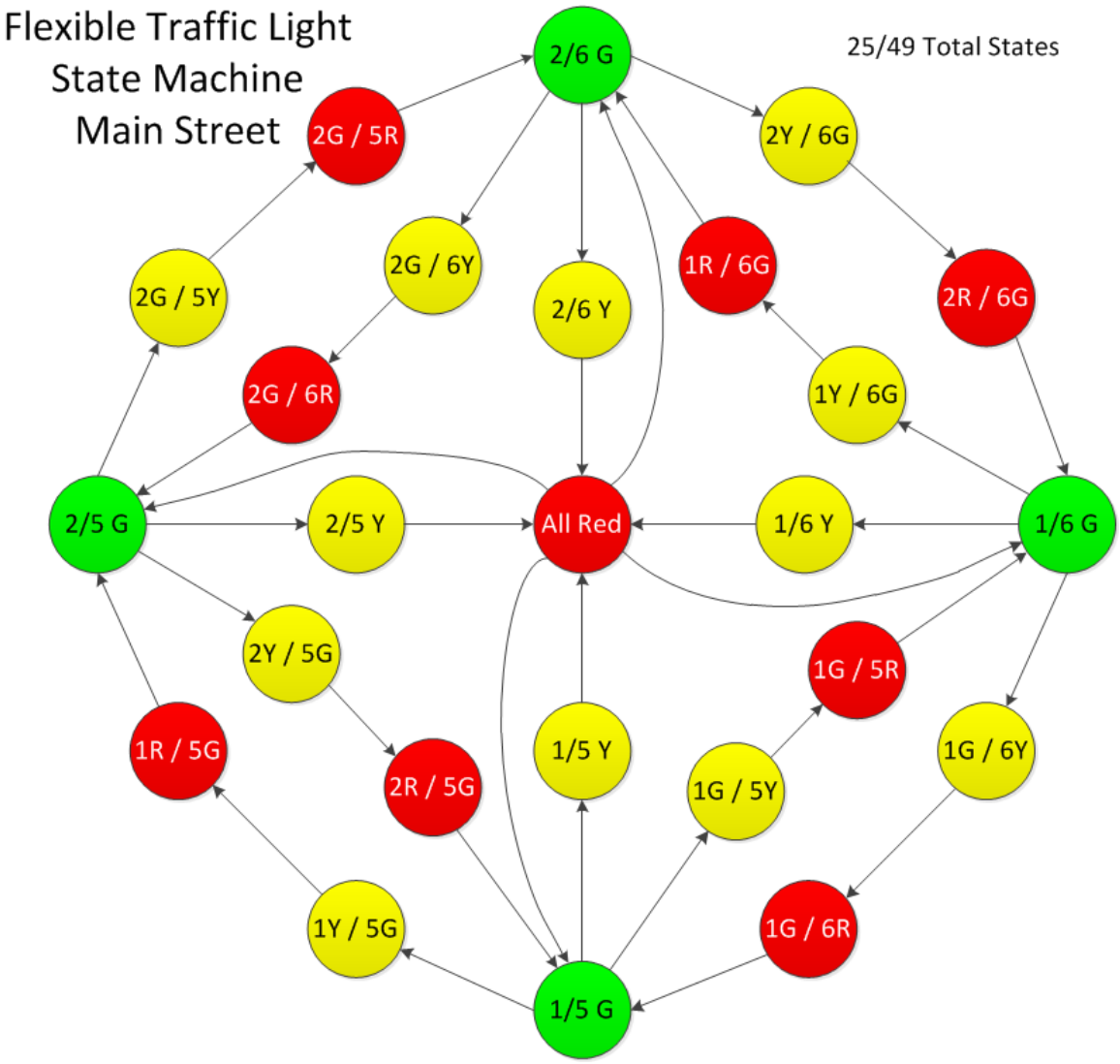

Figure 81: Main street portion of Flexible Dual-Ring Controller, Finite State Machine Representation

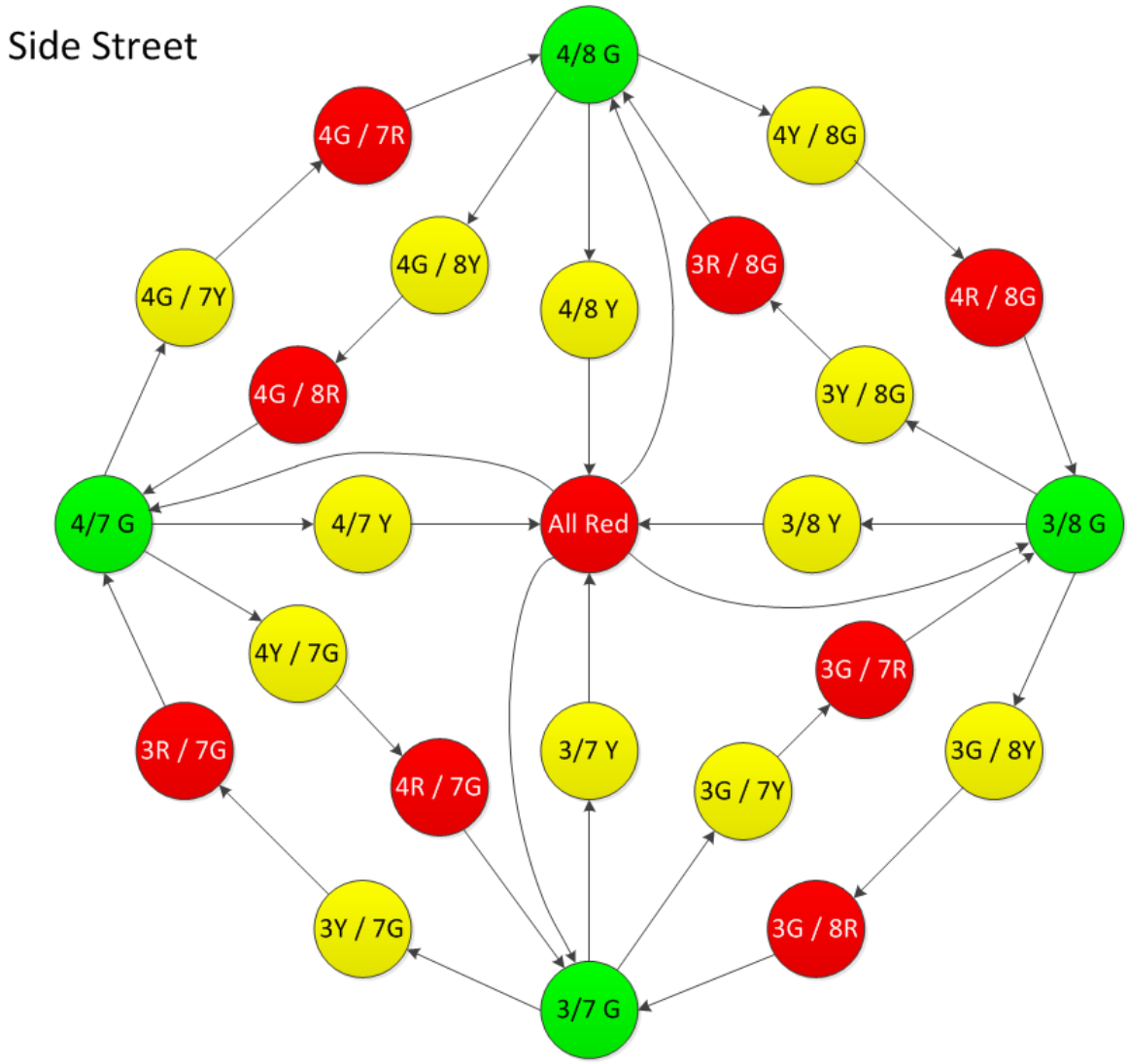

Figure 82: Side street portion of Flexible Dual-Ring Controller, Finite State Machine Representation

The total of 49 states allow for a variety of signal strategies to be implemented by the IMA, including "green extension," "early green," "phase insertion," and "phase rotation." Furthermore, the diagram shown in Figure 80 also indicates state transition information. At any given state, the set of possible next states is fully specified. In summary, the proposed flexible traffic light state machine provides a convenient

framework for visualizing adaptive signal control and providing state transition information to the IMA.

5.1.1.1.4 Signal Timing Optimization and MOE Selection

Using the flexible traffic light state machine presented in the previous section, signal phase duration and sequence may be optimized by the IMA to implement any specified MOE. Although MOEs may be easily compared in a simulation environment, additional factors must be considered for field deployment and system structure. Additional factors for consideration include the accuracy, accessibility, and privacy of information, computational complexity, and where the MOE falls on the scale of proactiveness and reactiveness. The accuracy of information is of vital importance for optimizing signal timing. Some MOEs, such as the travel time MOE, make use of predicted information to determine how overall travel times of individual vehicles might be impacted by potential signal phasing strategies. Any discrepancies between the predicted information and the eventual course or timing of events lead to sub-optimal performance of the signal optimizer. The accessibility, or ease of access, of information is an important consideration for physically implemented CV environment systems. Accessibility poses the practical question of whether a connected vehicle can obtain, package, and transmit the desired information in a timely manner. An example of an MOE with potentially poor accessibility is an MOE that relies on real-time vehicle emissions information. An issue which is increasingly gaining attention is the privacy and security of information in CV environments. Since vehicles would be potentially

transmitting detailed state information, the concern is that a connected vehicle could be tracked, or even worse hacked. The issue of computational complexity restricts MOE selection to MOEs that are mathematically tractable, and can be operated in a physical system in real-time. Finally, the consideration of proactivity versus reactivity is a system design issue. Reactivity is defined as an intersection merely responding to existing state information. Proactivity is defined as utilizing existing state information to predict future state information as an input into the signal optimization. An entirely reactive MOE has the benefit of using accurate information, but may fall behind in terms of providing the appropriate signal timing phases at the optimal time. In contrast, a completely proactive MOE has the benefit of staying ahead of current traffic conditions, but may be compromised by inaccurate predictions. As a result, an effective MOE strikes a balance on the scale of proactiveness and reactiveness.

Based on the above considerations, a number of MOEs including travel time, current delay, and queue length, and their variants, were explored in the process of selecting an appropriate MOE for a CV environment. Ultimately, a variation of queue length was selected as the most appropriate MOE for real-time signal optimization in a CV environment. The MOE of queue length satisfies all of the considerations listed in the preceding paragraph. For example, the queue length MOE is based on obtaining information on whether a vehicle is within range of an intersection and whether its speed is less than a maximum speed threshold. As a result, the queue length MOE relies on information that is 100% accurate, (whether a vehicle is in range of an intersection), and information that is easily accessible by vehicles (vehicle speed). Another advantage of

the queue length MOE is that vehicles are able to maintain privacy because they do not need to be tracked by the intersection. In terms of computational complexity, the queue length MOE is one of the simplest and most attractive MOEs for use in real-time signal optimization. Finally, on the scale of proactiveness and reactiveness, queue length generally falls closer to the reactive portion of the scale. However, increasing the maximum speed threshold under which vehicles are defined as being queued can move the queue length MOE closer to the center of the scale. For example, in a purely reactive queue length scheme, the maximum speed under which vehicles are queued is set to 0 mph, and only vehicles completely at rest will be served by a given signal phase. In contrast, a partially proactive queue length scheme considers vehicles which are about to stop as also being queued, which removes the constraint that vehicles must be completely stopped before being served by the intersection. Consequently, based on the advantages listed above, the results presented in subsequent sections utilize queue length as the MOE to evaluate the effectiveness of the proposed agent-based online adaptive signal control strategy.

Numerous variations of queue length optimization exist; therefore, a description of the exact queue length optimizer implemented follows, as shown in Figure 83. Queued vehicles were defined as vehicles within the communication radius of the IMA which had a velocity less than a user-defined threshold (e.g., 10 mph), and were approaching the intersection. The diagram in Figure 83 corresponds to the red block presented in Figure 76.

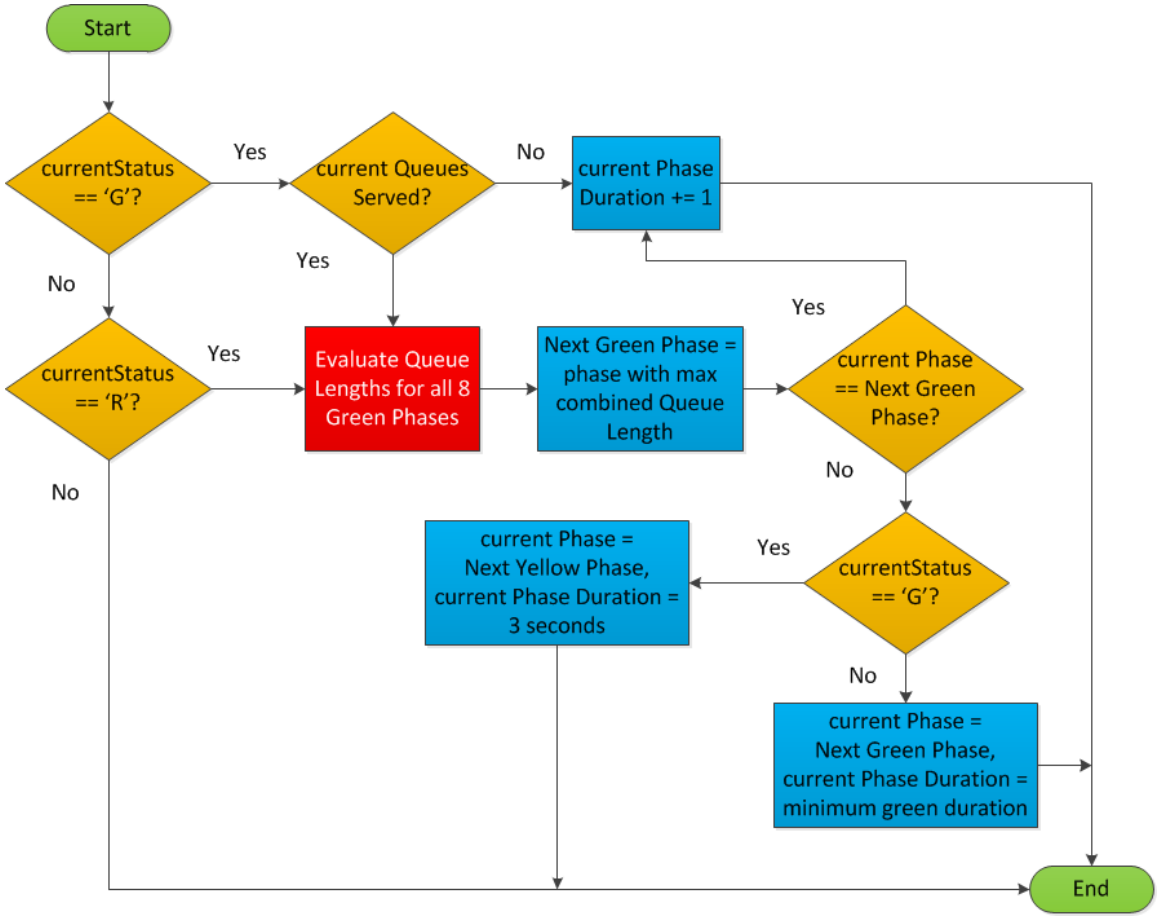

Figure 83: CV MAS Level 2 Diagram: Queue Length Signal Optimizer

The essential idea of the proposed queue length optimizer is to maximize the number of vehicles which are being served with a green light at the intersection. Recall from the IMA flow chart presented in Figure 76, that the signal optimizer is only called after a green phase has exceeded its minimum green allotment, or if the "All Red" state is expired. These two conditions are denoted in Figure 83 as the current state of the traffic light being either "G" or "R," respectively.

If the state is "R," the optimizer evaluates all eight possible green states to find the state with the maximum combined queue length across all lanes of the selected

movements. Once the optimal next state is calculated, the current phase is set to the selected green state, and is assigned the minimum green duration. In addition, all vehicles on any lane of the selected phases are internally marked as being currently served. The rationale behind keeping track of which vehicles are currently served is to allow the queues to fully discharge and avoid the undesirable "partial queue discharge" effect. The "partial queue discharge" effect occurs when a signal controller switches phases because the queue lengths on the currently served phases decrease (due to being currently served) to the point that a different phase combination has a larger combined queue length. The effect is undesirable because it leads to multiple stops for vehicles being served, and increases the loss time due to frequently switching phases.

After transitioning from a red colored state (Figure 80), the state becomes "G," and at the end of the minimum green duration, the IMA checks if all of the vehicles originally marked as being currently served have passed the stop bar. If not, the current green phase is repeatedly extended in one second increments until all of the marked vehicles have passed into the intersection. The proposed approach has the advantage of being able to switch to a green phase without having to predetermine its duration. Since additional vehicles may enter the currently served phase during the discharge of the marked queues, the possibility of remaining on the same green phase after the current queues have been served is permitted.

One issue that arises with the use of MOEs such as queue length for signal optimization is "green starvation." Green starvation occurs when certain approaches to

an intersection consistently have lower traffic volumes than other approaches. For example, if there is only one vehicle turning left from a minor street onto a major street, the traffic signal may prefer to keep serving the busier major street instead of switching to the minor street. In this case, although the IMA would be optimizing overall system performance by ignoring the single vehicle, the notion of fairness must be introduced. A single vehicle should not have to wait several minutes in order to be served by the intersection. One solution to the problem of green starvation is to modify the queue length MOE to incorporate information about the time elapsed since a particular signal phase was last served. If the time elapsed since a particular signal phase was last served is relatively high, then the queue length on that phase is weighted higher than the queue length on a signal phases that was more recently served. Essentially, the queue length is multiplied by "aging" factors. The relationship between the time elapsed since a signal phase was last served and the value of the multiplicative aging factors can be adjusted based on individual localities needs. For the results shown in the following sections, the relationship between the time elapsed since a signal phase was last served and the value of the multiplicative aging factor was set to be a quadratic equation fitting the points (0, 1), (30, 2), and (120, 10), where x is the input (elapsed time in seconds), and y is the output (aging factor value). The first point corresponds to the queue length remaining unmodified if the signal phase was just served. The second point corresponds to the queue length being weighted twice as high as normal if half a minute has elapsed since the signal phase was last served. Finally, the third point corresponds to the queue length being weighted 10 times higher than normal if a full two minutes has elapsed since the

signal phase was last served. A plot of the quadratic equation is shown in Figure 84 below.

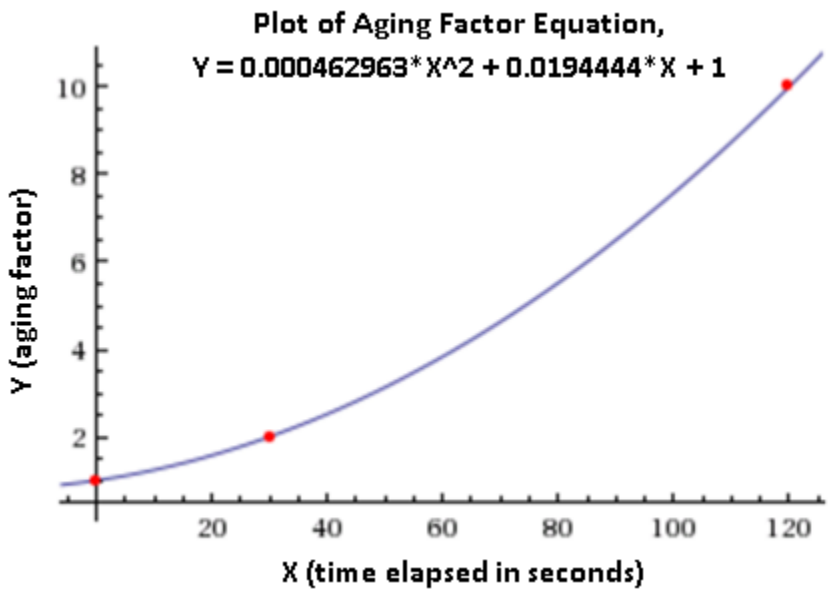

Figure 84: Plot of Quadratic Aging Factor Equation

5.1.1.2 Simulation Setup

In order to implement the adaptive CV signal optimization strategy in simulation, PARAMICS 6.9.3 was selected. In addition, EPA's MOVES software was integrated in order to provide information on the environmental performance metrics. Detailed information regarding the software setup may be found in chapter 3. The adaptive CV signal optimization strategy was tested in comparison to several baseline strategies for various sensitivity analyses. The first baseline simulated was an intersection with fixed phase signal timing where the cycle length was fixed at 120 seconds. The second baseline simulated used a cycle length calculated using the unmodified Webster's

formula for cycle length. The unmodified equation for Webster's cycle length, C, is $C = (1.5 * L + 5)/(1 - CS/S)$, where L is the loss time in the cycle due to the duration of yellow and red signal phases, where CS is the sum of the critical lane volumes over every signal phase for the intersection, and where S is the saturation flow rate. Once the cycle length is determined, the signal splits are determined based on the ratios of critical volumes for each phase. A third baseline was also implemented for the demand profile sensitivity analysis, where cycle length is calculated using the HCM method. In terms of general simulation parameters, the speed limit for each intersection was set at 45 mph. Each simulation run was conducted for 1 hour, with additional time to permit all vehicles to exit the simulation.

5.1.1.3 PARAMICS Network Description

The isolated intersection used to evaluate the adaptive CV signal optimizer and the baseline signal control strategies is shown in Figure 85 below. The adaptive signal control version of the intersection used PARAMICS movement priorities in order to fully control the intersection movements. A second version of the intersection for testing the baseline signal control strategies used PARAMICS built-in signal control module. Nevertheless, the physical layout of the intersection remained identical for both versions of the network. The intersections were designed to have 4 approaches, with 3 lanes each, plus a left-turn bay of 500 feet. The overall dimensions of the intersection were 2414 feet by 2414 feet. The turning movements for the lanes are shown in Figure 86 below. From left to right, the left-most lane was set as an exclusive left-turn lane, the middle two lanes

were set as through-only lanes, and the right-most lane was set as a shared through and right-turn lane with a right-turn on red policy. As shown in the upper right portion of Figure 85, a GUI was developed to indicate which signal phase was active, and to display the queue lengths of each phase in real-time. Every signal phase starts at a minimum of 8 seconds, and is extended as necessary to fully clear the queue of vehicles being served. The upper left portion of Figure 85 includes a red box which indicates the current maximum length phase during the simulation.

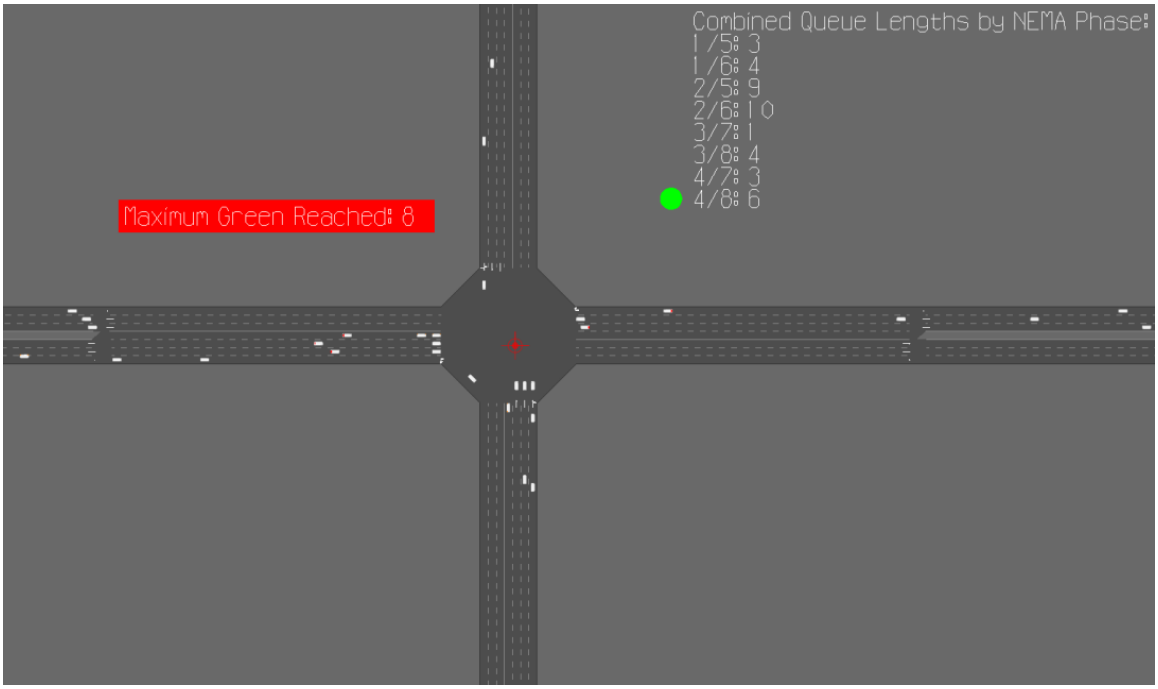

Figure 85: Isolated Intersection PARAMICS Network

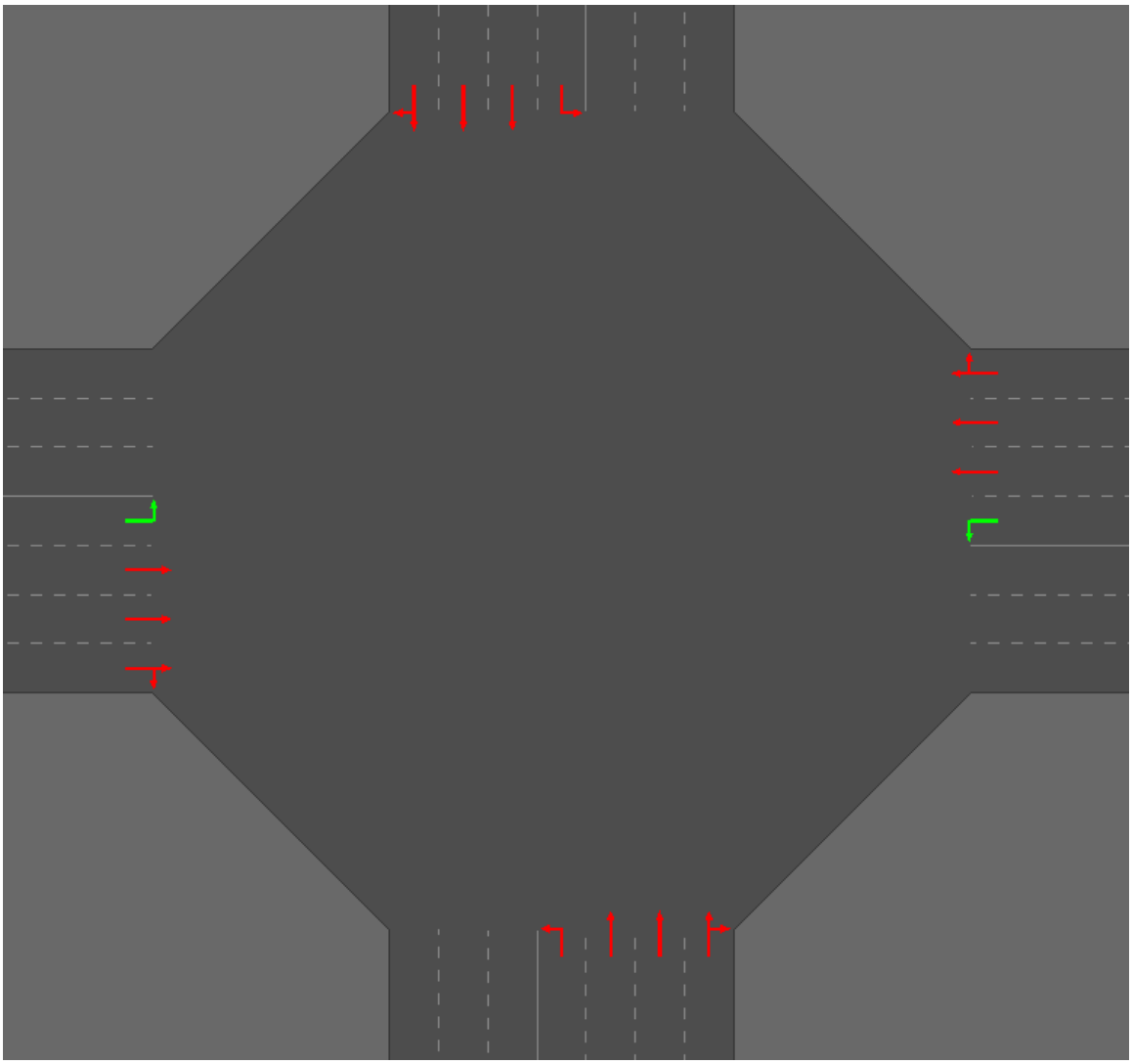

Figure 86: Turning Movements for Isolated Intersection

5.1.1.4 Volume Sensitivity Analysis

For the volume sensitivity analysis, a series of traffic volumes ranging from 1000 vehicle per hour to 6000 vehicles per hour in 500 vehicles per hour increments was tested. The overall traffic on the major street was set to be 50% higher than the traffic on the minor street, and the turning ratios for left-turn movement, through movement, and

right-turn movements was set to 20%, 70%, and 10%, respectively. In addition, a constant demand profile over the course of the one hour simulation was used.

5.1.1.4.1 Connected Vehicle Queue Length Signal Optimization versus Fixed Phase Signal Timing

The CV queue length optimizer is given no information regarding incoming traffic. In contrast, the signal splits for the fixed phase signal timing intersection are determined based on complete knowledge of the origins and destinations of the incoming traffic. The assumption of *a priori* information being available to the fixed phase signal timing intersection is equivalent to the signal timing being perfectly tuned. The results for the CV queue length signal optimizer relative to fixed phase signal timing are shown in Figures 87-91 and Tables 33-35. The travel time savings are highest at low traffic volumes, and gradually decrease as traffic volume is increased, until the time savings are erased at 6000 vehicles per hour. An identical trend may be observed in terms of energy saved by using CV queue length signal optimization instead of fixed phase signal timing. The energy benefits are highest at low traffic volumes, and decrease as the traffic volume increases. The emissions savings ranged primarily from -5% to 15%, with the greatest savings occurring at low traffic volumes. The emissions savings are positive at the lower volumes because the intersection is able to more quickly respond to incoming vehicles as opposed to vehicles which may have to wait at a fixed phase signal the better part of a 120 second cycle in order to be served regardless of the absence of vehicles on other signal phases. The emissions savings are slightly negative at high traffic volumes due to

an increase in the number of vehicle stops. The increase in the number of vehicle stops is due to the intersection beginning to reach its capacity.

Due to the addition of the dedicated left-turn bay, the capacity of the intersection is around 6500 vehicles per hour. The results indicate at near saturated conditions, adaptive signal control does not provide additional benefits over fixed phase signal timing. Generally, fixed phase signal timing is considered a relatively weak baseline; however, under near saturated and saturated conditions, fixed phase signal timing performs better than adaptive signal control strategies.

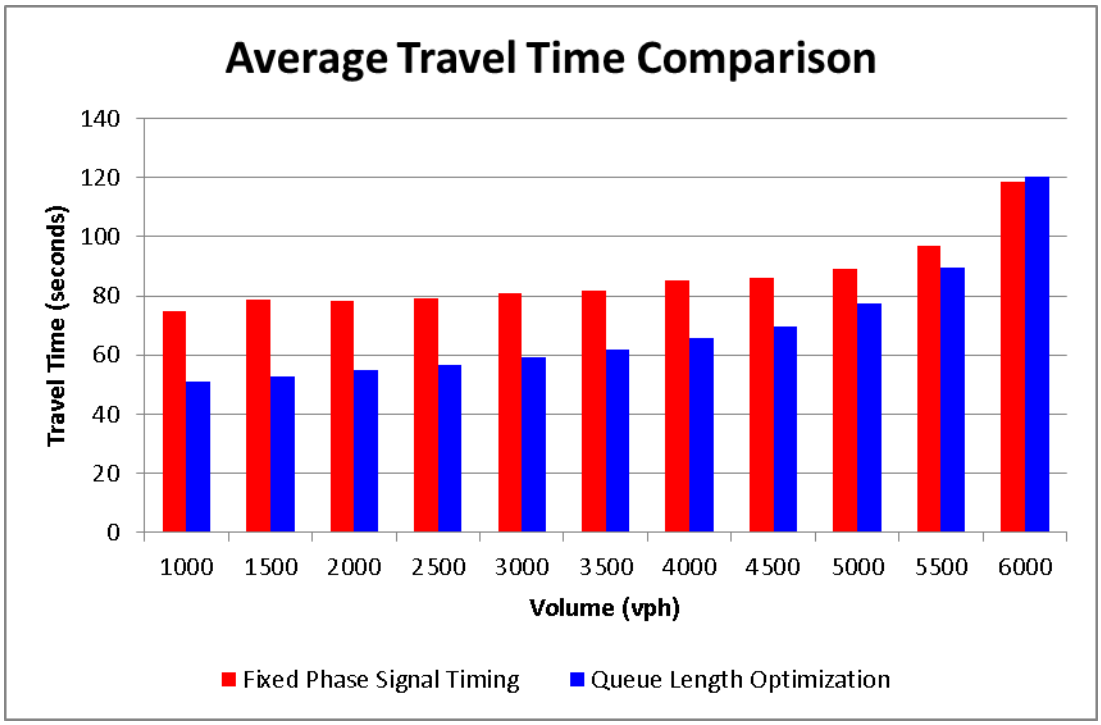

Figure 87: Average Travel Time Comparison of CV Queue Length Signal Optimization and Fixed Phase Signal Timing for an Isolated Intersection

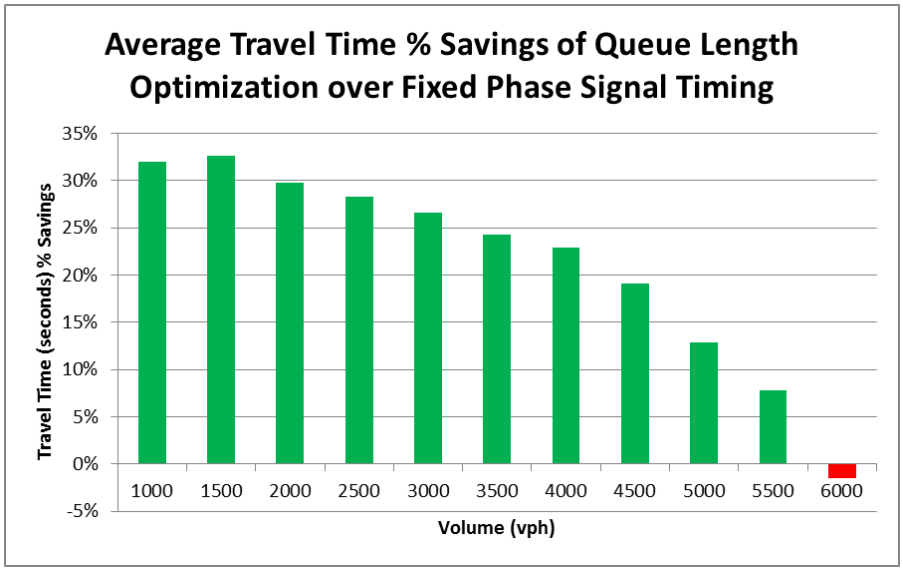

Figure 88: Average Travel Time Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection

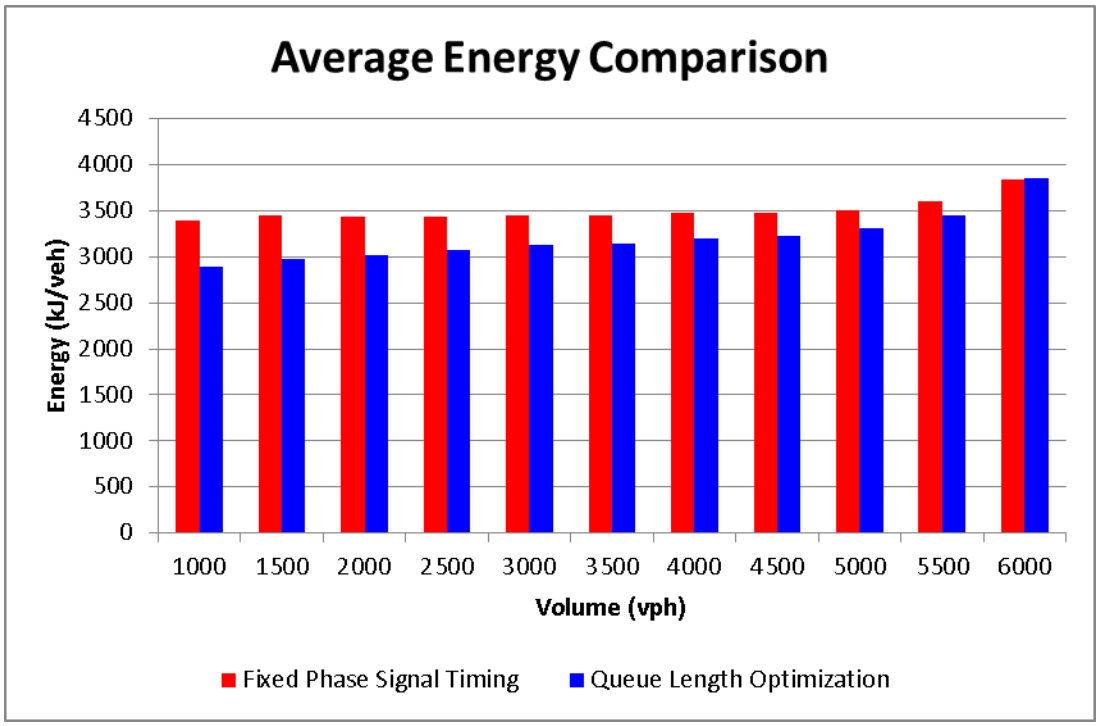

Figure 89: Average Energy Comparison Of CV Queue Length Signal Optimization and Fixed Phase Signal Timing for an Isolated Intersection

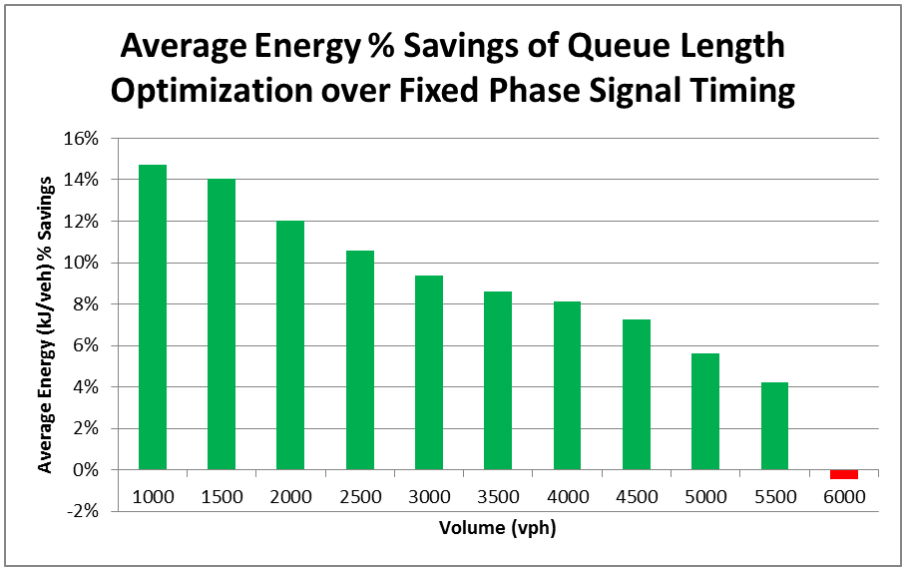

Figure 90: Average Energy Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection

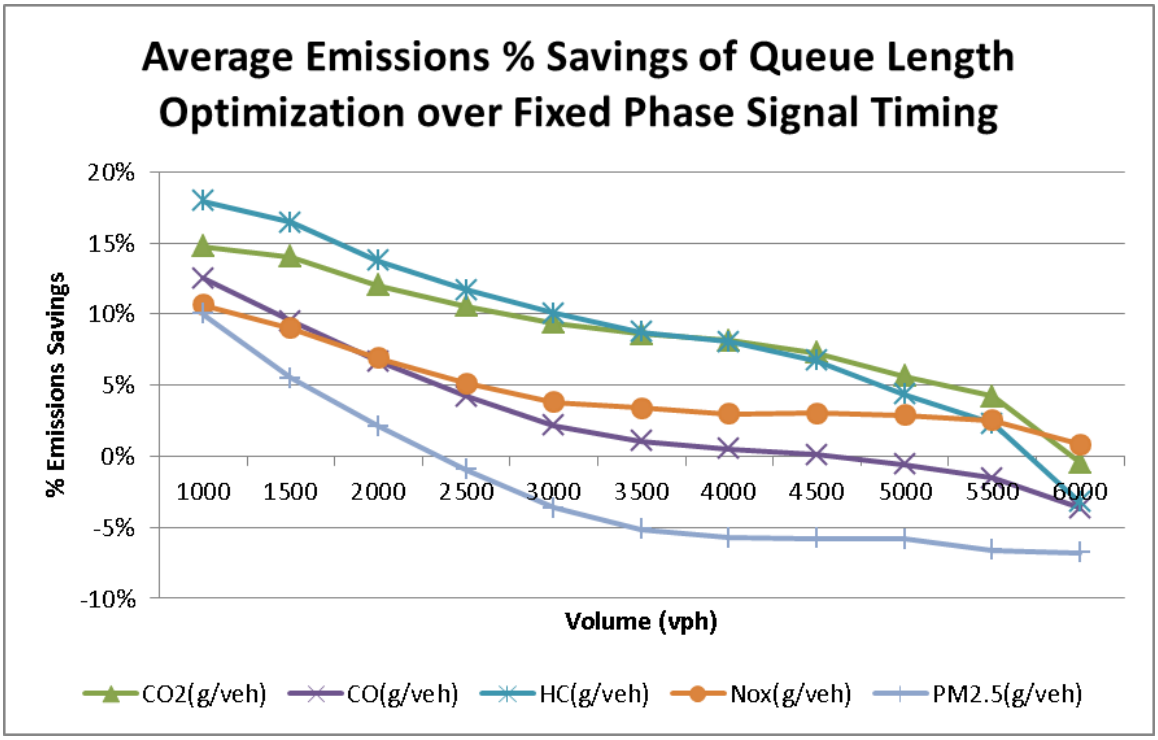

Figure 91: Average Emissions Percent Savings of CV Queue Length Signal Optimization over Fixed Phase Signal Timing on an Isolated Intersection

Table 33: Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	3388.3116	243.5169	5.3228	0.1662	0.5759	0.0389	74.8485
1500	3455.2288	248.3262	5.3119	0.1689	0.5814	0.0382	78.5087
2000	3430.3610	246.5389	5.2031	0.1666	0.5747	0.0371	78.2152
2500	3436.9265	247.0108	5.1667	0.1665	0.5732	0.0367	78.9594
3000	3448.2000	247.8210	5.0976	0.1666	0.5705	0.0358	80.8024
3500	3443.3450	247.4721	5.0291	0.1660	0.5660	0.0350	81.7712
4000	3483.9148	250.3878	5.0226	0.1682	0.5661	0.0346	85.2737
4500	3482.0136	250.2512	4.9640	0.1678	0.5625	0.0339	86.2439
5000	3506.8703	252.0376	4.9296	0.1691	0.5606	0.0333	89.0625
5500	3597.5330	258.5535	4.9426	0.1747	0.5613	0.0327	97.0571
6000	3840.3389	276.0037	5.0622	0.1894	0.5704	0.0323	118.7739

Table 34: CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	2888.9053	207.6249	4.6571	0.1364	0.5145	0.0350	50.8897
1500	2970.3667	213.4795	4.8056	0.1411	0.5289	0.0361	52.9012
2000	3017.3586	216.8567	4.8555	0.1437	0.5349	0.0364	54.9200
2500	3073.5148	220.8926	4.9482	0.1470	0.5438	0.0370	56.6369
3000	3124.9574	224.5898	4.9870	0.1499	0.5486	0.0371	59.3266
3500	3147.3108	226.1963	4.9760	0.1515	0.5467	0.0368	61.9434
4000	3200.2195	229.9988	4.9977	0.1546	0.5491	0.0366	65.7862
4500	3228.6623	232.0430	4.9585	0.1565	0.5453	0.0359	69.8067
5000	3309.1727	237.8292	4.9577	0.1617	0.5444	0.0352	77.5807
5500	3445.4696	247.6247	5.0186	0.1707	0.5472	0.0349	89.4644
6000	3857.5189	277.2384	5.2460	0.1956	0.5657	0.0345	120.4958

Table 35: % Improvement of CV Queue Length Optimization over Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	14.74%	14.74%	12.51%	17.96%	10.66%	10.00%	32.01%
1500	14.03%	14.03%	9.53%	16.44%	9.03%	5.48%	32.62%
2000	12.04%	12.04%	6.68%	13.74%	6.93%	2.10%	29.78%
2500	10.57%	10.57%	4.23%	11.72%	5.13%	-0.96%	28.27%
3000	9.37%	9.37%	2.17%	10.04%	3.82%	-3.63%	26.58%
3500	8.60%	8.60%	1.06%	8.71%	3.41%	-5.13%	24.25%
4000	8.14%	8.14%	0.50%	8.06%	3.00%	-5.74%	22.85%
4500	7.28%	7.28%	0.11%	6.72%	3.05%	-5.78%	19.06%
5000	5.64%	5.64%	-0.57%	4.37%	2.90%	-5.84%	12.89%
5500	4.23%	4.23%	-1.54%	2.30%	2.50%	-6.59%	7.82%
6000	-0.45%	-0.45%	-3.63%	-3.24%	0.82%	-6.80%	-1.45%

5.1.1.4.2 Connected Vehicle Queue Length Signal Optimization versus Webster Signal Timing

The cycle length for the Webster signal timing intersection, as determined by the Webster equation, was applied individually to each traffic volume in order to provide a strong baseline. The CV queue length optimizer is given no information regarding the incoming volume of traffic. However, the Webster signal timing intersection is given information not only on the total volume of incoming traffic, but also which lanes the overall origins and destinations of the incoming vehicles. Accordingly, the signal splits for the Webster signal timing intersection are determined using the OD matrix. In contrast, the signal splits for the CV queue length signal optimization are determined in real-time without any use of *a priori* information.

The results for the queue length CV signal optimizer relative to volume-specific Webster signal timing are shown in Figures 92-96 and Tables 36-38. The range of

average travel time savings for the CV queue length signal optimization over Webster signal timing falls between -5% and 13%. Previously, the range of average travel time savings for the CV queue length signal optimization over fixed phase signal timing was shown to be between -1% and 32%. The average travel time savings are lower relative to Webster signal timing due to the relative strength of the baseline. In this case, Webster signal timing is a stronger baseline than fixed phase signal timing due to the cycle length being set independently for each traffic volume for the Webster signal timing. As was the case with the comparison with fixed phase signal timing, the average travel time benefits are highest at the lowest traffic volumes. The average travel time savings are negative for traffic volumes greater than or equal to 5000 vehicles per hour. The average energy savings ranged from 0% to about 10%, with the higher range of benefits occurring at the lower traffic volumes. The average emissions savings ranged from -2% to 13%, with the higher range of benefits also occurring at low traffic volumes. In contrast to the comparison with the fixed phase signal timing, the emissions savings are predominantly positive across the tested traffic volumes. The reason for the additional positive savings is due to the difference in cycle lengths between the baselines. The Webster signal timing used cycle lengths that were much shorter than the cycle length of 120 seconds used for the fixed phase signal timing baseline. One of the potential disadvantages of using a shorter cycle length is that the number of vehicle stops increases. An increase in the number of vehicle stops may be shown to be correlated to an increase in vehicle emissions.

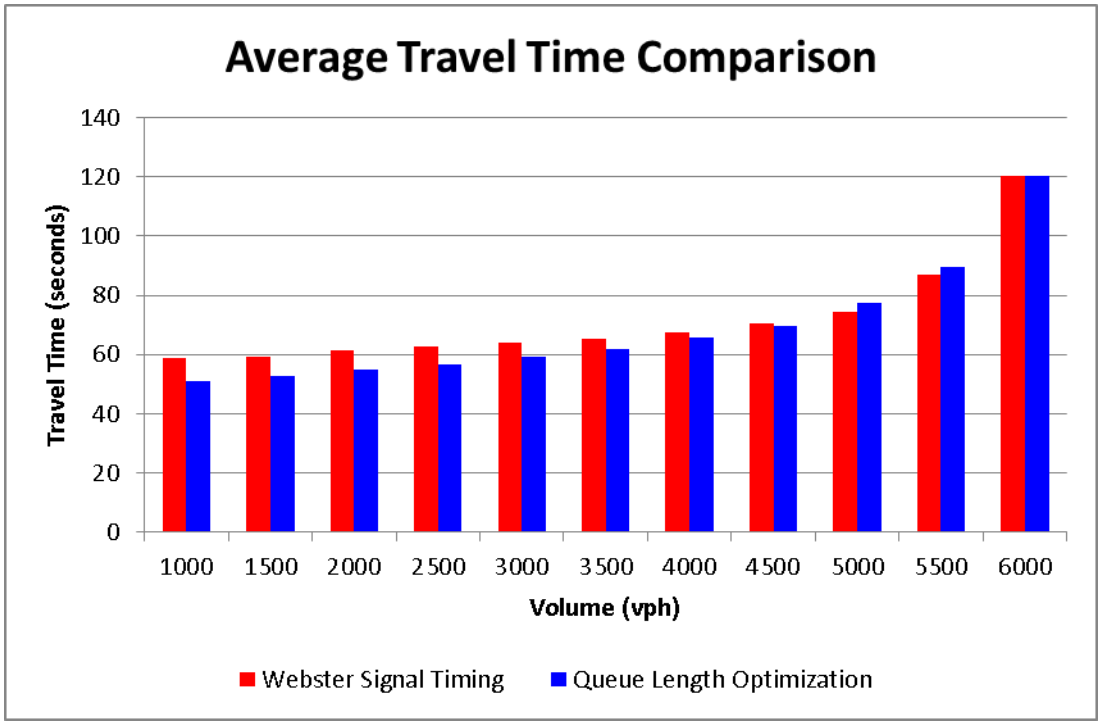

Figure 92: Average Travel Time Comparison of CV Queue Length Signal Optimization and Webster Signal Timing for an Isolated Intersection

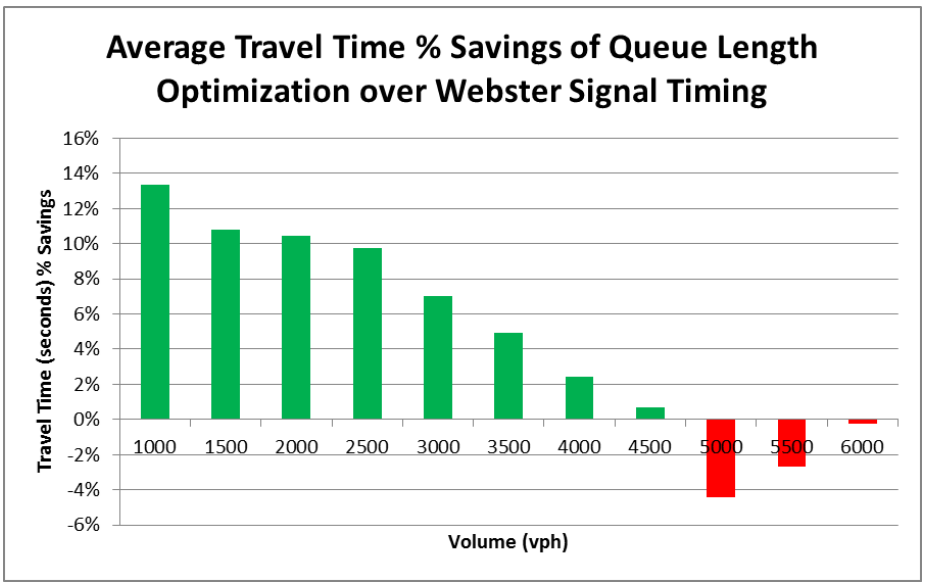

Figure 93: Average Travel Time Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection

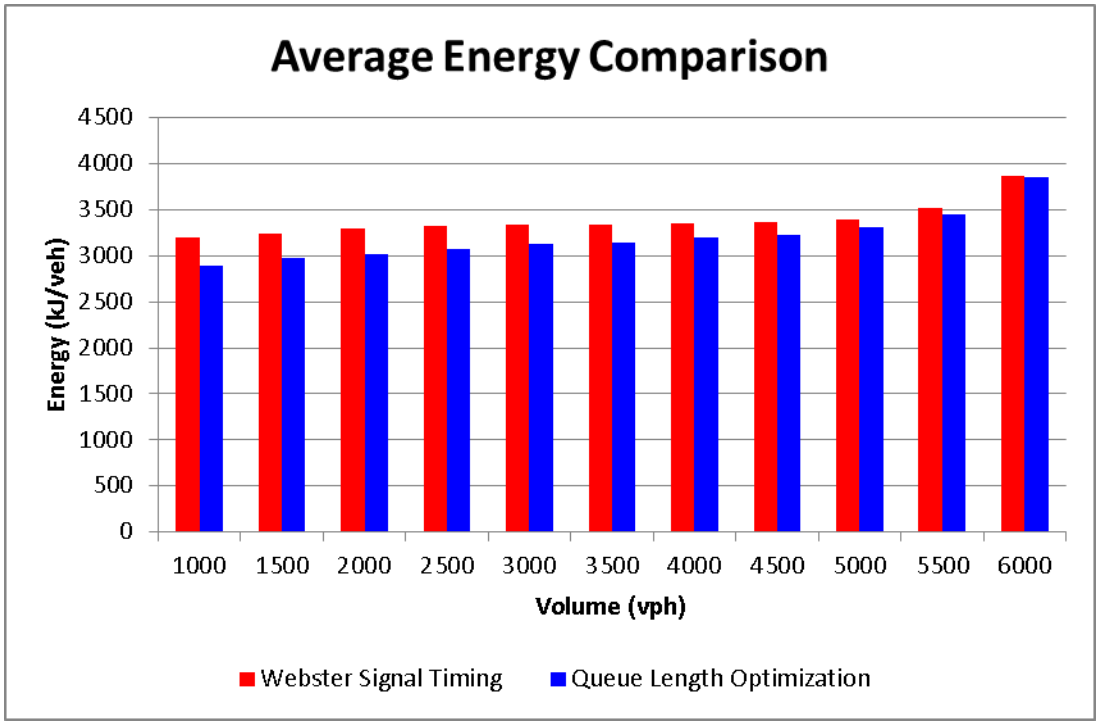

Figure 94: Average Energy Comparison of CV Queue Length Signal Optimization and Webster Signal Timing for an Isolated Intersection

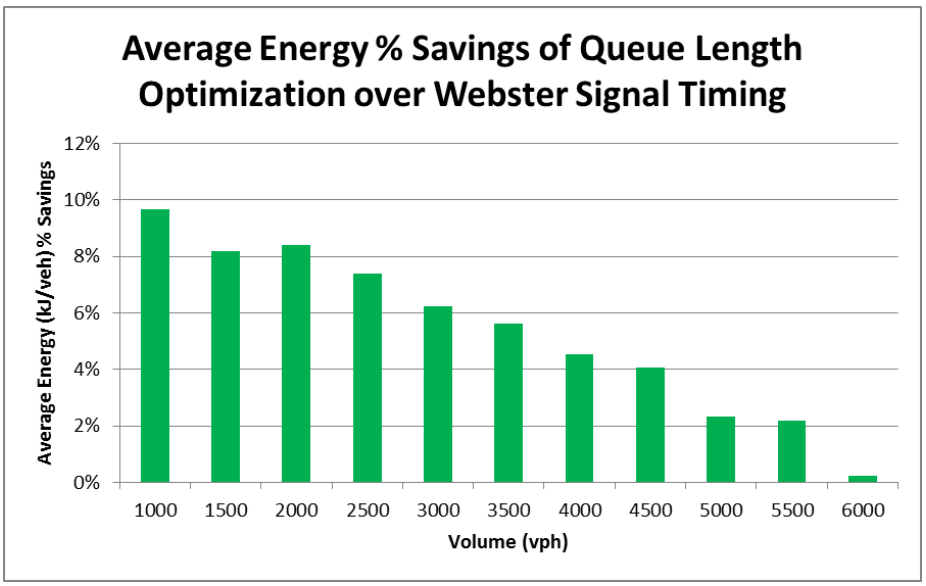

Figure 95: Average Energy Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection

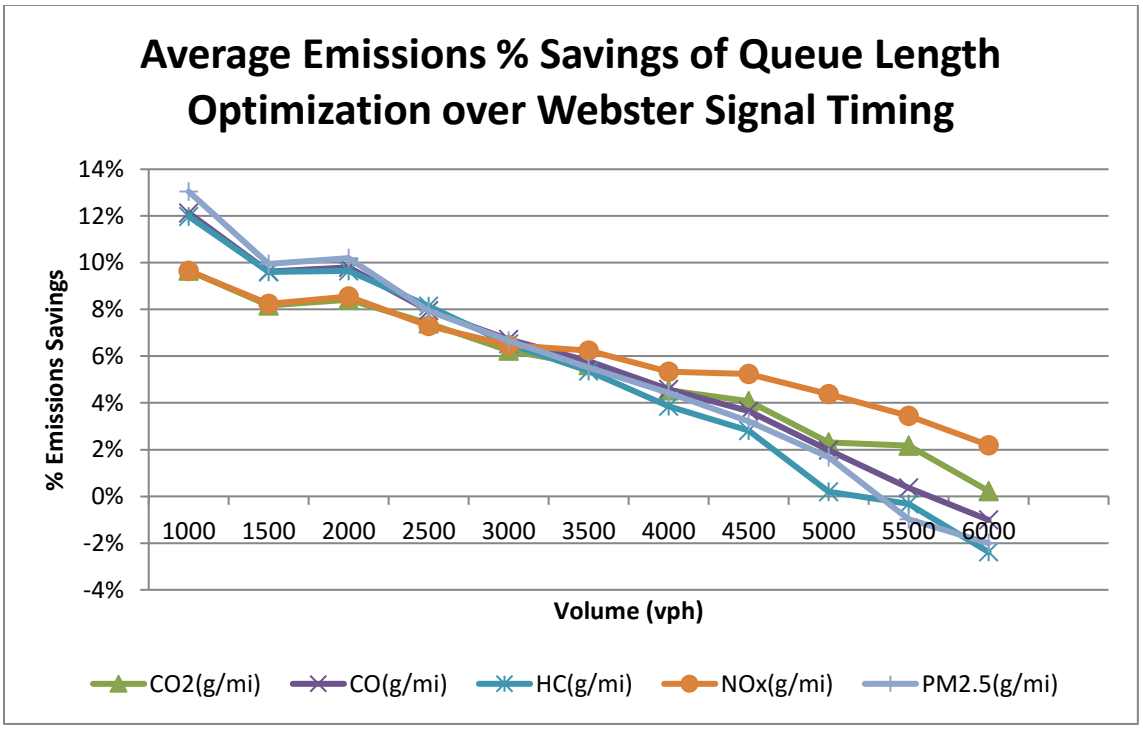

Figure 96: Average Emissions Percent Savings of CV Queue Length Signal Optimization over Webster Signal Timing on an Isolated Intersection

Table 36: Webster Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	3197.8442	229.8281	5.3001	0.1549	0.5694	0.0402	58.7392
1500	3234.6362	232.4724	5.3175	0.1561	0.5764	0.0401	59.2864
2000	3294.6454	236.7852	5.3830	0.1591	0.5849	0.0405	61.3116
2500	3319.0298	238.5377	5.3760	0.1601	0.5866	0.0402	62.7683
3000	3332.7369	239.5228	5.3459	0.1604	0.5866	0.0397	63.7971
3500	3335.0394	239.6883	5.2817	0.1601	0.5831	0.0389	65.1727
4000	3352.5423	240.9462	5.2377	0.1608	0.5800	0.0383	67.4273
4500	3365.7337	241.8943	5.1466	0.1611	0.5755	0.0371	70.2974
5000	3387.5355	243.4611	5.0575	0.1620	0.5693	0.0358	74.2839
5500	3522.0480	253.1284	5.0368	0.1701	0.5668	0.0346	87.1314
6000	3866.3465	277.8729	5.1936	0.1910	0.5784	0.0338	120.2226

Table 37: CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	2888.9053	207.6249	4.6571	0.1364	0.5145	0.0350	50.8897
1500	2970.3667	213.4795	4.8056	0.1411	0.5289	0.0361	52.9012
2000	3017.3586	216.8567	4.8555	0.1437	0.5349	0.0364	54.9200
2500	3073.5148	220.8926	4.9482	0.1470	0.5438	0.0370	56.6369
3000	3124.9574	224.5898	4.9870	0.1499	0.5486	0.0371	59.3266
3500	3147.3108	226.1963	4.9760	0.1515	0.5467	0.0368	61.9434
4000	3200.2195	229.9988	4.9977	0.1546	0.5491	0.0366	65.7862
4500	3228.6623	232.0430	4.9585	0.1565	0.5453	0.0359	69.8067
5000	3309.1727	237.8292	4.9577	0.1617	0.5444	0.0352	77.5807
5500	3445.4696	247.6247	5.0186	0.1707	0.5472	0.0349	89.4644
6000	3857.5189	277.2384	5.2460	0.1956	0.5657	0.0345	120.4958

Table 38: % Improvement of CV Queue Length Optimization over Webster Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	9.66%	9.66%	12.13%	11.98%	9.65%	13.04%	13.36%
1500	8.17%	8.17%	9.63%	9.60%	8.24%	9.95%	10.77%
2000	8.42%	8.42%	9.80%	9.64%	8.55%	10.20%	10.42%
2500	7.40%	7.40%	7.96%	8.15%	7.29%	7.93%	9.77%
3000	6.23%	6.23%	6.71%	6.52%	6.47%	6.66%	7.01%
3500	5.63%	5.63%	5.79%	5.36%	6.24%	5.51%	4.96%
4000	4.54%	4.54%	4.58%	3.85%	5.33%	4.43%	2.43%
4500	4.07%	4.07%	3.65%	2.81%	5.24%	3.21%	0.70%
5000	2.31%	2.31%	1.97%	0.20%	4.37%	1.68%	-4.44%
5500	2.17%	2.17%	0.36%	-0.32%	3.44%	-0.97%	-2.68%
6000	0.23%	0.23%	-1.01%	-2.38%	2.19%	-2.01%	-0.23%

5.1.1.5 Demand Profile Sensitivity Analysis

An additional sensitivity analysis was performed on the demand profile of traffic arriving at the intersections. A demand profile is a sequence that can modify an OD

matrix to provide additional traffic during specified time intervals. For example, a sequence such as {25, 25, 25, 25} specifies that 25% of the traffic for a given OD pair should be released by the simulator during the first quarter of the simulation time period. A demand profile of {100} leaves the OD matrix unmodified. For the demand profile sensitivity test, the volume to capacity ratio was set to 0.5 for each signal control strategy, and demand profile sequences with standard deviations of 1, 3, and 5 were tested relative to a demand profile of {100}. The specific demand profile sequences are shown in Table 39. The individual numbers in the demand profile sequences with non-zero standard deviation specify the percentage of the overall hourly volume for OD pairs for specific 5-minute intervals. The first number in the sequence specifies the first 5-minute interval during the hour-long simulation run. Although the overall hourly volume is set to 0.5 times the V/C ratio, the 5-minute interval volumes each have their own V/C ratio. The individual sequence numbers were constrained to ensure that the 5-minute interval V/C ratios did not exceed 1. The signal splits for the fixed phase signal timing were set with the assumption that the ratios of traffic utilizing each signal phase were known perfectly *a priori*. Likewise, both the cycle length and the signal splits for the HCM and Webster signal timing were set with perfect *a priori* knowledge. The signal timing for the CV queue length signal optimization was not based on the availability of the OD matrix, and was instead calculated in real-time during the simulation.

The relative percent sensitivity results are shown in Figure 97 and Table 40 below, where the results are measured relative to a demand profile with a standard deviation of 0. The fixed phase signal timing strategy was the least sensitive to increases

in the variation of the demand profile. The HCM signal timing strategy exhibited the highest relative sensitivity at the highest standard deviation tested. An additional method of analyzing sensitivity is to observe the absolute values of travel times for different demand profiles. Accordingly, absolute travel time sensitivity results are shown in Figure 98 and Table 41 below. When viewed through the perspective of absolute travel times, it becomes evident that the CV queue length signal optimization strategy has the lowest travel time across all of the demand profiles tested. Although the fixed phase signal timing was the least sensitive in terms of relative percent sensitivity, fixed phase signal timing was generally the worst in terms of absolute travel time.

Table 39: Demand Profile Sequences used for Demand Profile Sensitivity Test

Standard Deviation of Demand Profile	Demand Profile Sequence
0	{100}
1	{8, 8, 7, 8, 9, 8, 10, 7, 9, 8, 10, 8}
3	{3, 9, 7, 12, 6, 14, 8, 7, 7, 11, 9, 7}
5	{3, 12, 4, 11, 13, 17, 5, 7, 3, 3, 14, 8}

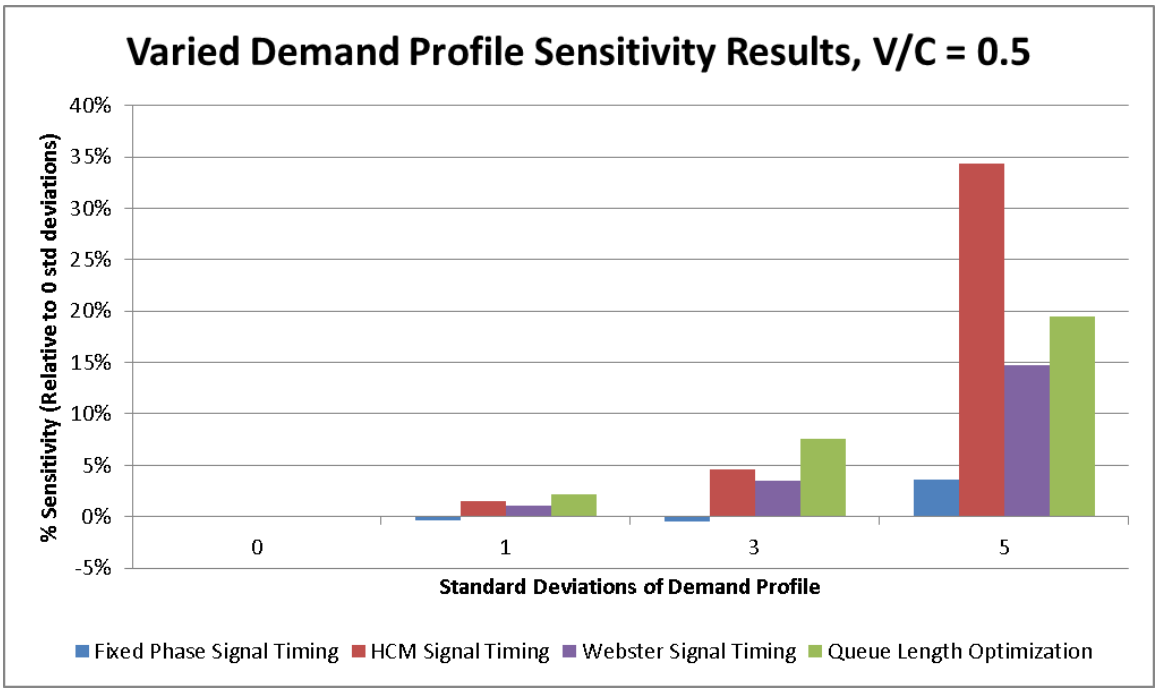

Figure 97: Percent Travel Time Sensitivity to Varied Demand Profile

Table 40: Percent Travel Time Sensitivity to Varied Demand Profile

		Signal Timing Method			
		Fixed Phase	HCM	Webster	CV Queue Length Optimization
Standard Deviations of Demand Profile	0	---	---	---	---
	1	0%	1%	1%	2%
	3	0%	5%	3%	8%
	5	4%	34%	15%	19%

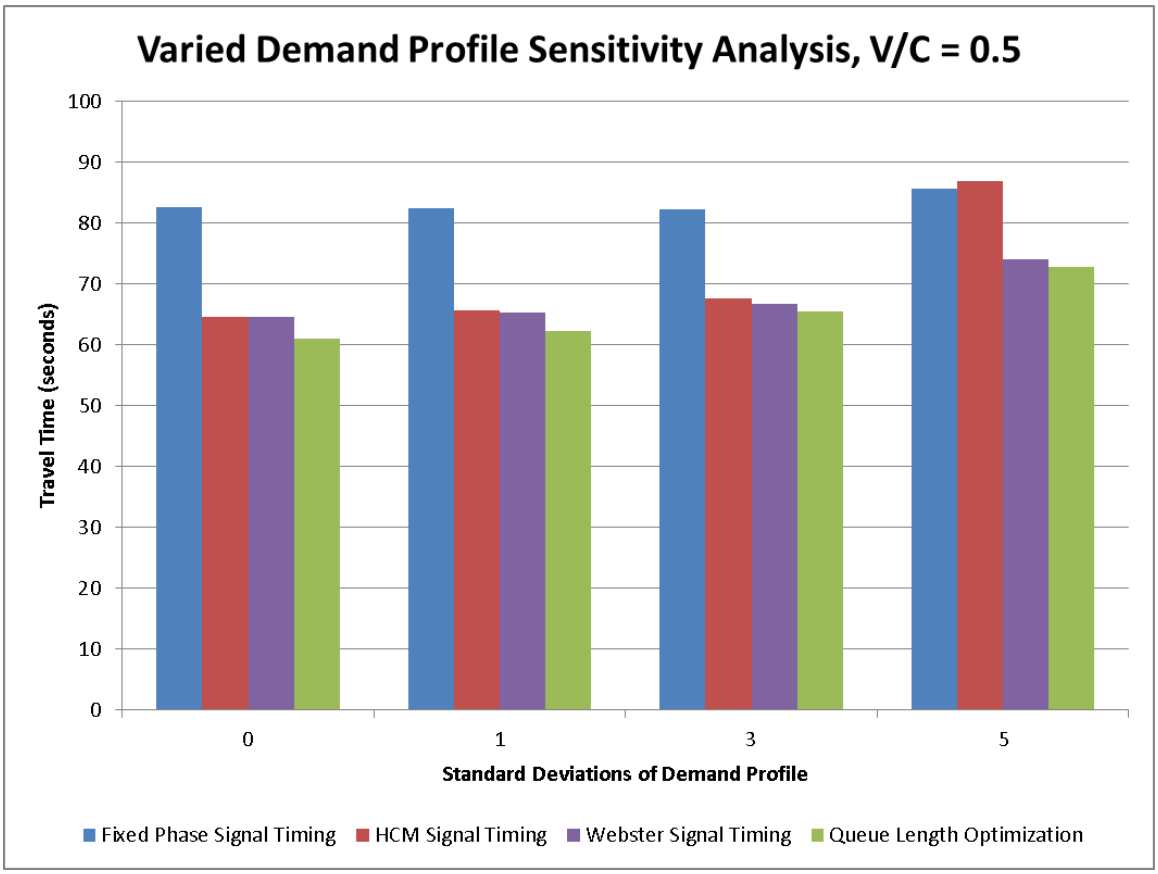

Figure 98: Travel Time Sensitivity to Varied Demand Profile

Table 41: Travel Time Sensitivity to Varied Demand Profile

		Signal Timing Method			
		Fixed Phase	HCM	Webster	CV Queue Length Optimization
Standard Deviations of Demand Profile	0	82.6888	64.6436	64.5542	60.9619
	1	82.3683	65.5899	65.2621	62.2486
	3	82.2838	67.6122	66.8010	65.5581
	5	85.6539	86.8335	74.0670	72.8055

5.1.2 Corridor-Level Connected Vehicle Signal Optimization

5.1.2.1 Application Description

The CV queue length signal optimizer described in section 5.1.1 on an isolated intersection may readily be extended to multiple intersections. A sequence of intersections may be referred to as a signalized corridor. Traditionally, signalized corridors are operated using a coordinated fixed phase signal timing where the intersections share the same signal timing plan with time offsets based on the physical distance between subsequent intersections. The coordinated fixed phase signal timing plan is designed to permit vehicles traveling along the corridor to be able to travel through multiple intersections without stopping. In the case of adaptive signal control, a given intersections signal plan is unfixed. Consequently, one method for extending the CV queue length signal optimizer from an isolated intersection to a corridor of intersections is to apply the same optimizer to each intersection. Each intersection is set to operate independently of adjacent intersections, constituting what may be referred to as decentralized corridor management. The following sections will describe the implementation, testing, and results of simulating a decentralized signalized corridor in a CV environment.

5.1.2.2 Simulation Setup

As with the case of the isolated intersection, PARAMICS 6.9.3 was also used to simulate a corridor of 3 signalized intersections. The PARAMICS API provided access to mobility results, and EPA's MOVES provided emissions results. Further information

regarding the software setup may be found in chapter 3. The OD matrix for a corridor of three intersections was set such that each intersection would retain the same level of traffic as the single isolated intersection described in section 5.1.1.3. Instead of using fixed turning ratios and a fixed ratio of major to minor street traffic, a custom OD matrix generator was developed to allow these values to vary every 5 minutes to better reflect the variations in real-world traffic. In addition, a demand profile with a standard deviation of 1 was used to further emulate real-world traffic. Accordingly the ratio of major street to minor street traffic was set to 1.5 with a standard deviation of 0.2. The percentage of left-turn movement traffic was set to 20% with a standard deviation of 0.05. Likewise, the percentage of through movement traffic was set to 70% with a standard deviation of 0.05. All remaining traffic, roughly 10%, was set to be right-turn movement traffic. In addition, the ratio of traffic originating from the north to traffic originating from the south was set to 1.05 with a standard deviation of 0.1. Likewise, the ratio of traffic originating from the west to traffic originating from the east was set to 1.05 with a standard deviation of 0.1. Using the input values mentioned above a set of 12 OD matrices were generated, one for every 5 minute interval. Each simulation run was conducted for 1 hour with additional time for vehicles to clear the network.

5.1.2.3 PARAMICS Network Description

The PARAMICS network for a corridor of 3 intersections is shown in Figure 99 below. Each intersection is identical to the isolated intersection described in section 5.1.1.3. The distance between the stop bars of successive intersections was set to 2415

feet. The speed limit throughout the network was set at 45 mph. Based on the distance between intersections and the speed limit, the progression time, (the time a vehicle takes to get from one intersection to the next), was calculated to be 37 seconds. For the baseline coordinated fixed phase signal timing network, the cycle length which permits the largest "green window" for coordinating East-West and West-East traffic was 74 seconds. The phrase "green window" refers to the time duration allotted during a cycle to coordinated movements between multiple intersections. Based on a complete knowledge of the OD matrix, the effective "green window," (the coordinated green phase duration plus 2 seconds of yellow), was set to 18 seconds out of the 74 second cycle. The left and right intersections depicted in Figure 99 operate with a time offset of 0 seconds. The center intersection shown in the Figure operates with a time offset of 37 seconds. A time-space diagram, shown in Figure 100, summarizes the baseline coordinated fixed phase signal timing plan for the 3-intersection corridor. The decentralized CV queue length signal optimization network operated without the use of a predetermined signal timing plan. Each intersection was permitted to determine its own signal timing based on the vehicles within range of the given intersection. The communication radius for each intersection, (~600 feet), was set to fully overlap the beginning of the left-turn bays for the purpose of the intersection being able to distinguish between left-turn movement traffic and through movement traffic. If the communication radius is set shorter, then the IMA is less informed in its optimization of signal timing. If the communication radius is expanded beyond the length of the left-turn bay, then vehicles are required to

communicate their turning intentions to the IMA. Transmitting turning intentions may be viewed as a violation of driver privacy.

Figure 99: 3-Intersection Corridor PARAMICS Network

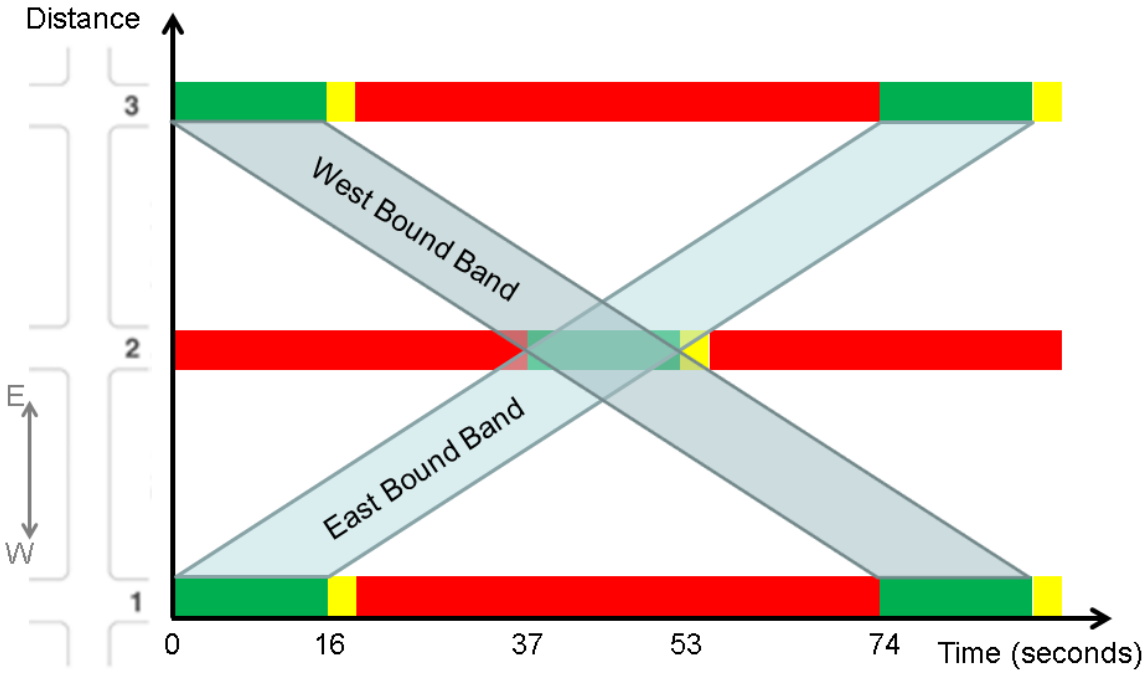

Figure 100: Time-Space Diagram showing Coordinated Fixed Phase Signal Timing Plan for PARAMICS Network (see previous Figure)

5.1.2.4 Volume Sensitivity Analysis

For the volume sensitivity analysis, a series of traffic volumes ranging from 1000 vehicle per hour per intersection (vphpi) to 6000 vphpi in 500 vphpi increments was tested. The average corridor level results are shown in Figures 101-105 and Tables 42-44. The decentralized CV queue length signal optimizer outperforms the coordinated fixed phase signal timing for traffic volumes less than or equal to 4000 vphpi. The maximum average travel time savings of 19% was achieved at the lowest volume tested (1000 vphpi). Similarly, the maximum average energy savings of nearly 8% was achieved at the same volume. As the traffic volume was increased, average travel time, energy, and emissions savings decreased. At traffic volumes greater than 4000 vphpi, the average travel time and energy savings were negative, reaching minimums of -22% and -8%, respectively. Emissions savings were for the most part negative, varying predominantly between -10% and +10%, with the positive savings occurring at the low traffic volumes.

Additional insight can be gained by dividing the results into the categories of coordinated-phase vehicles and uncoordinated-phase vehicles. Coordinated-phase vehicles are defined as vehicles which travel the full length of the corridor. Examining coordinated-phase vehicle statistics helps determine if, and to what extent, coordinated-phase vehicles are negatively impacted by passing through independently adaptive intersections instead of progressing through a coordinated fixed phase signal timing corridor. The coordinated-phase vehicle results are shown in Figures 106-110 and Tables

45-47, and are followed by uncoordinated-phase vehicle results which are shown in Figures 111-115 and Tables 48-50. A comparison of the average results, the coordinated-phase vehicle results, and the uncoordinated-phase vehicle results is shown in Figures 116-117.

As hypothesized, the use of decentralized adaptive signal control negatively impacted the average travel time, energy consumption, and emissions of vehicles traveling the full length of the signalized corridor. At the lowest traffic volume tested, (1000 vphpi), there is a small, (less than 2%), benefit in terms of travel time, energy consumption, and emissions. The reason for the small benefit is that vehicles operating under coordinated fixed phase signal timing must wait until the coordinated phase begins. Once the coordinated phase begins, vehicles are able to progress through the remaining two intersections with relatively little delay. In contrast, vehicles operating under decentralized adaptive signal timing experience a certain amount of delay at each of the three intersections. At the traffic volume of 1000 vphpi, the average delay experienced by vehicles passing through the three decentralized adaptive signal timing intersections was slightly less than the average delay experience by vehicles waiting for the coordinated phase to begin in the coordinated fixed phase signal timing corridor. For traffic volumes greater than 1000 vphpi, the average delay per intersection summed over the three intersections for the decentralized adaptive signal timing corridor outweighs the average delay experienced by vehicles waiting for the start of the coordinated phase in the coordinated fixed phase signal timing baseline corridor. The penalty experienced by

traffic traveling through the length of the corridor increases with volume and reaches a maximum of -59% in terms of travel time and -19% in terms of energy.

In contrast to the coordinated-phase vehicles, the uncoordinated-phase vehicles generally experience benefits under decentralized adaptive signal control relative to coordinated fixed phase signal timing. The maximum uncoordinated-phase vehicle benefits of 23% for travel time and 9% for energy occur at a traffic volume of 1000 vphpi. The benefits decrease with volume, remaining positive up to 4500 vphpi. The benefits are negative for traffic volumes greater than 4500 vphpi. The emissions savings for uncoordinated-phase vehicles are in the 0% to 10% range for traffic volumes less than or equal to 2500 vphpi. Examining the average, coordinated-phase, and uncoordinated-phase vehicle statistics reveals that the overall average is lowered by the relatively poor performance of coordinated-phase vehicles in the decentralized adaptive signal control corridor. However, the overall benefits are still positive for traffic volumes up to 4000 vphpi due to the positive benefits experienced by uncoordinated-phase vehicles.

The most significant contribution presented in the chapter is that CV adaptive signal control is superior to traditional adaptive signal control methods. In addition, queue length is an eminently suitable MOE for use with the newly developed CV adaptive signal control optimizer. In the context of an isolated intersection, the average maximum benefits of the new CV adaptive signal control optimizer versus a fixed phase intersection controller given *a priori* information, exceeded 13% and 10% for average travel time and average energy, respectively. In the case of a corridor of traffic

intersections, positive benefits were observed up to a traffic volume of 4000 vphpi, with the uncoordinated phase vehicles on average benefiting the most.

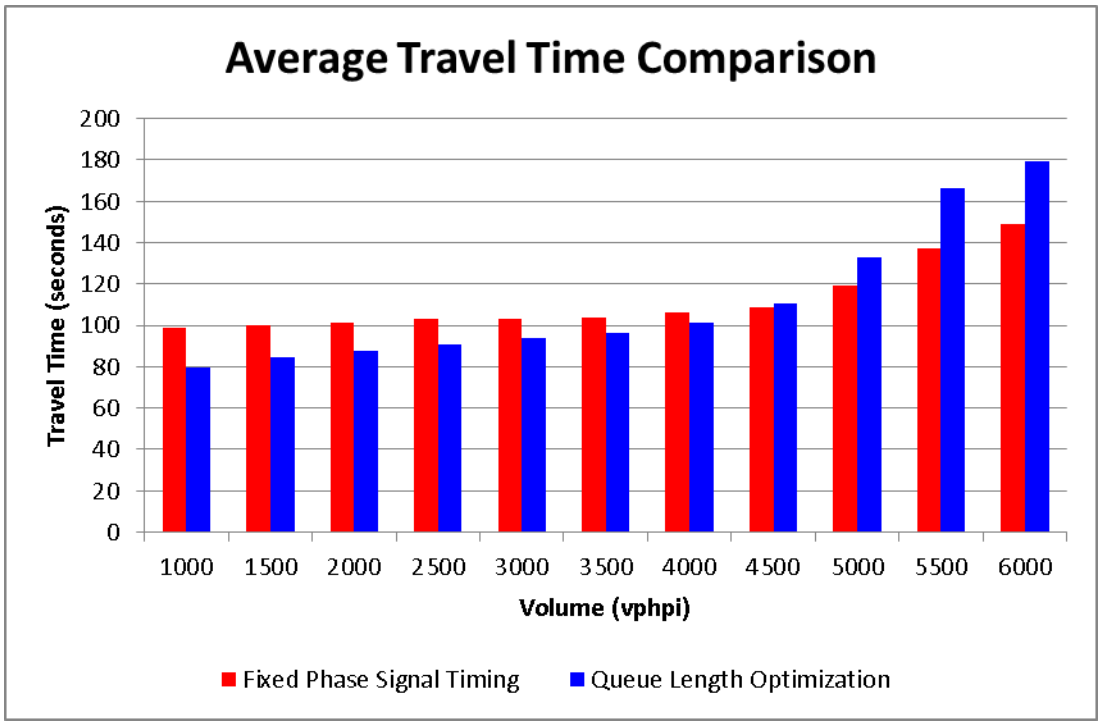

Figure 101: Average Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

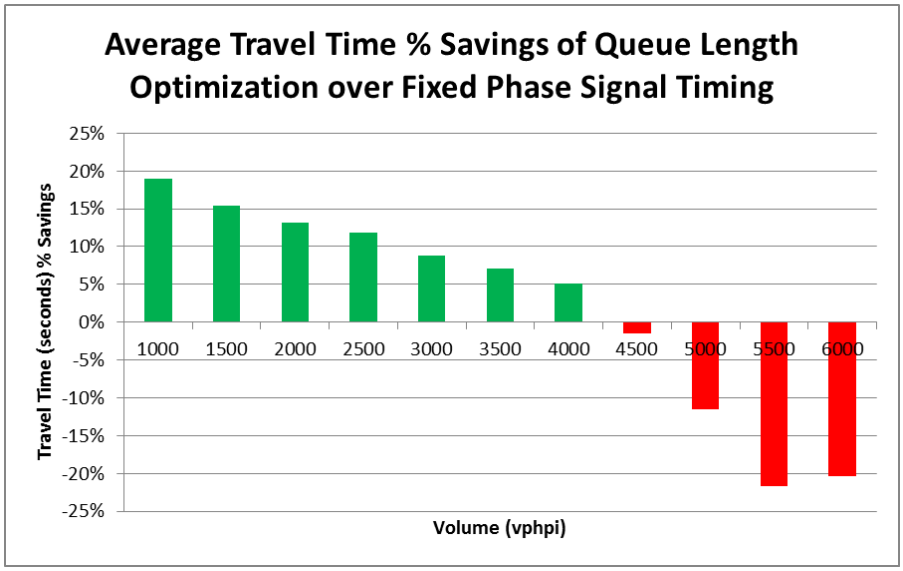

Figure 102: Average Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

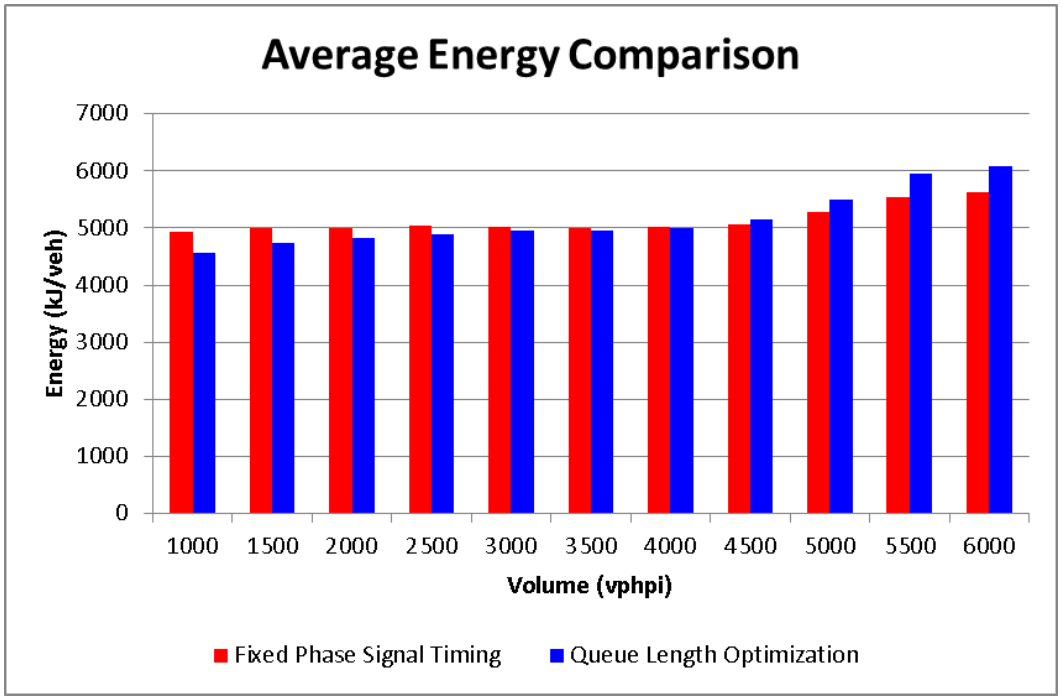

Figure 103: Average Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

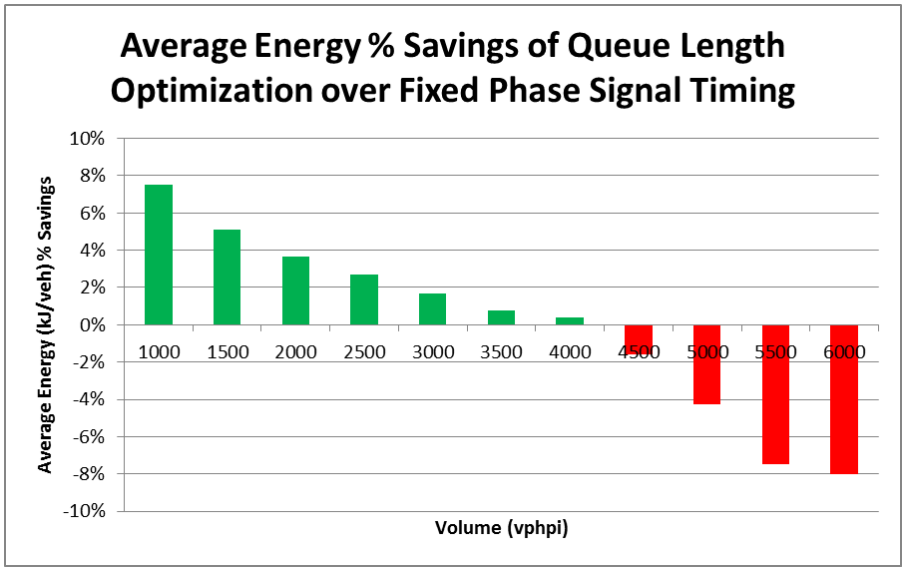

Figure 104: Average Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

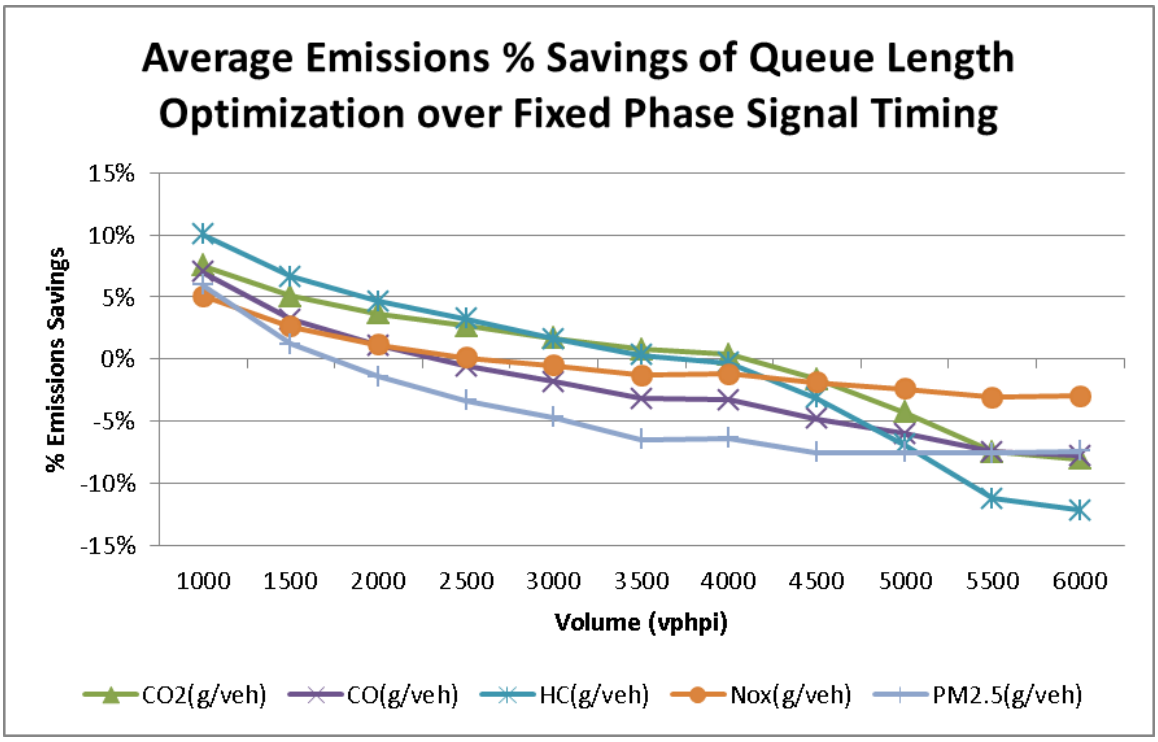

Figure 105: Average Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

Table 42: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	4925.0679	353.9632	7.3467	0.2324	0.8340	0.0525	98.6335
1500	5000.7984	359.4060	7.4100	0.2353	0.8462	0.0526	100.0019
2000	5003.7649	359.6192	7.3507	0.2351	0.8434	0.0519	101.0038
2500	5031.3857	361.6042	7.3500	0.2367	0.8442	0.0516	102.9197
3000	5026.5534	361.2569	7.3021	0.2360	0.8426	0.0510	102.8570
3500	4993.5554	358.8854	7.2024	0.2345	0.8332	0.0500	103.5022
4000	5015.0644	360.4312	7.2067	0.2365	0.8313	0.0499	106.4753
4500	5064.1490	363.9589	7.2211	0.2387	0.8356	0.0496	108.6725
5000	5275.9349	379.1798	7.4089	0.2503	0.8557	0.0503	119.1899
5500	5538.6240	398.0591	7.5681	0.2656	0.8707	0.0502	136.9902
6000	5623.3941	404.1515	7.5606	0.2707	0.8696	0.0495	149.1811

Table 43: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	4555.3127	327.3892	6.8316	0.2090	0.7916	0.0494	79.8905
1500	4746.6884	341.1433	7.1728	0.2197	0.8240	0.0520	84.6244
2000	4820.7921	346.4691	7.2716	0.2241	0.8337	0.0527	87.6669
2500	4896.2985	351.8957	7.3917	0.2290	0.8435	0.0534	90.6824
3000	4943.6683	355.3001	7.4348	0.2322	0.8469	0.0534	93.8211
3500	4955.5583	356.1546	7.4336	0.2338	0.8440	0.0532	96.1943
4000	4996.1249	359.0701	7.4459	0.2372	0.8414	0.0531	101.0808
4500	5145.1674	369.7817	7.5682	0.2462	0.8514	0.0534	110.2949
5000	5501.8461	395.4159	7.8531	0.2677	0.8761	0.0541	132.9384
5500	5951.9169	427.7622	8.1354	0.2955	0.8974	0.0540	166.6054
6000	6074.7658	436.5912	8.1514	0.3037	0.8955	0.0532	179.5833

Table 44: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	7.51%	7.51%	7.01%	10.04%	5.09%	5.95%	19.00%
1500	5.08%	5.08%	3.20%	6.62%	2.62%	1.21%	15.38%
2000	3.66%	3.66%	1.08%	4.68%	1.14%	-1.43%	13.20%
2500	2.68%	2.68%	-0.57%	3.27%	0.08%	-3.35%	11.89%
3000	1.65%	1.65%	-1.82%	1.63%	-0.51%	-4.68%	8.78%
3500	0.76%	0.76%	-3.21%	0.30%	-1.29%	-6.49%	7.06%
4000	0.38%	0.38%	-3.32%	-0.32%	-1.21%	-6.38%	5.07%
4500	-1.60%	-1.60%	-4.81%	-3.16%	-1.88%	-7.52%	-1.49%
5000	-4.28%	-4.28%	-6.00%	-6.93%	-2.39%	-7.54%	-11.53%
5500	-7.46%	-7.46%	-7.49%	-11.24%	-3.07%	-7.58%	-21.62%
6000	-8.03%	-8.03%	-7.81%	-12.21%	-2.98%	-7.45%	-20.38%

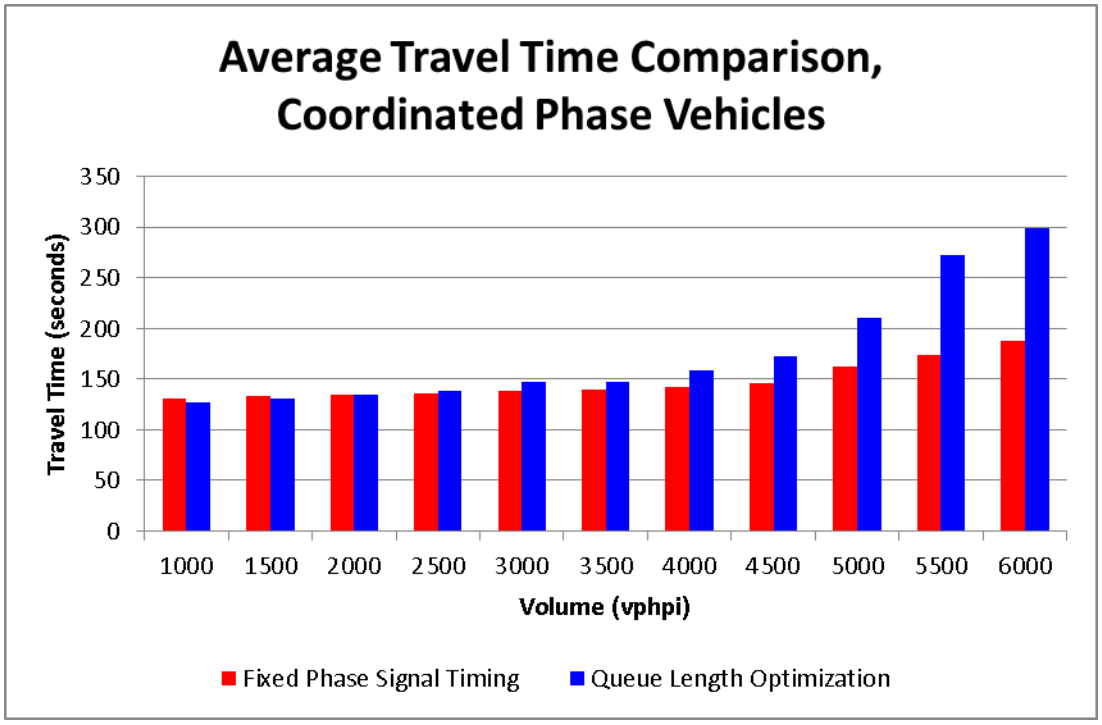

Figure 106: Coordinated Phase Vehicle Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

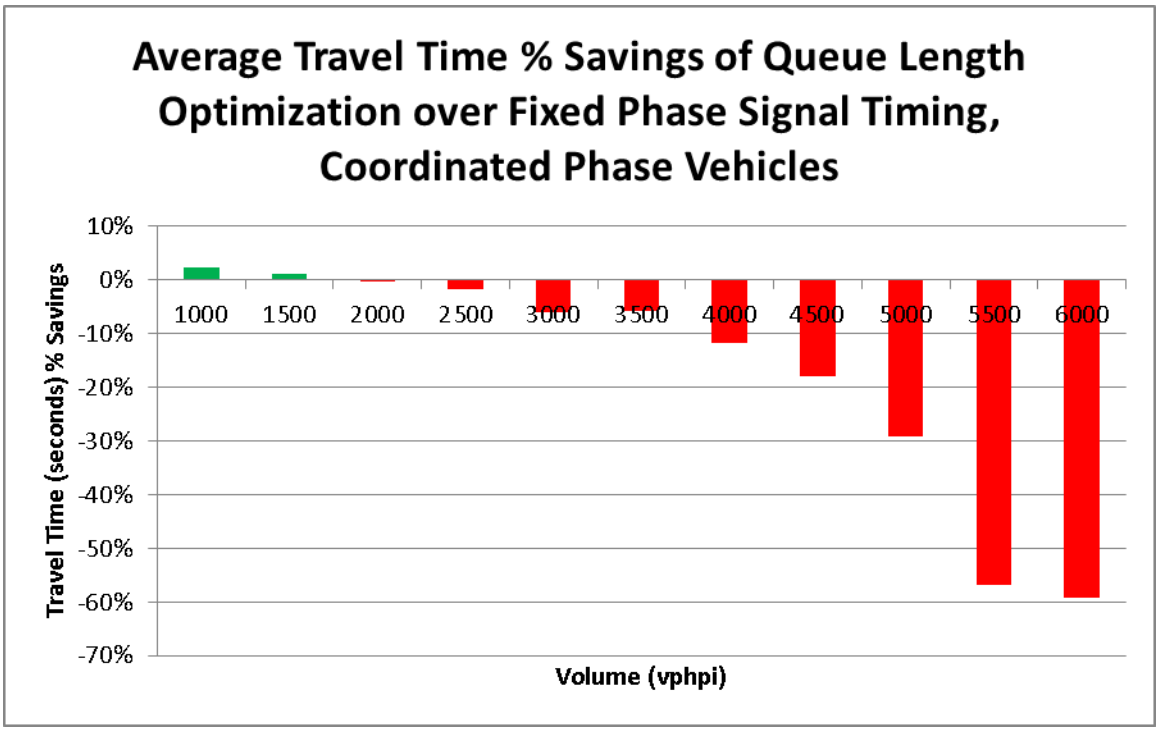

Figure 107: Coordinated Phase Vehicle Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

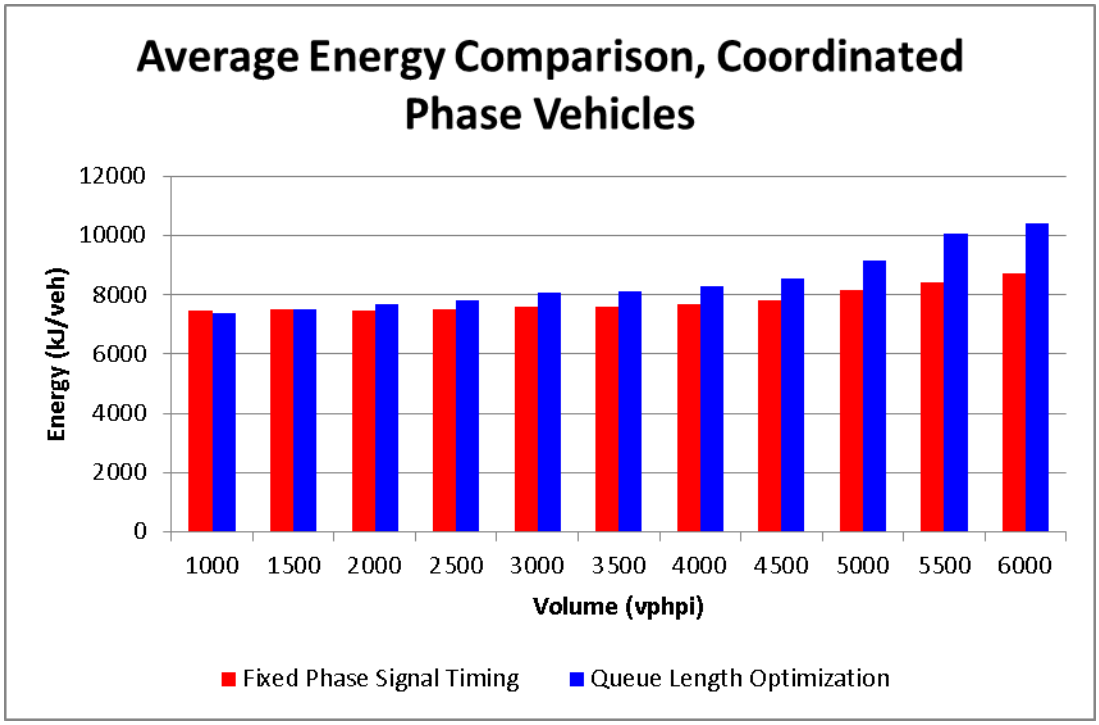

Figure 108: Coordinated Phase Vehicle Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

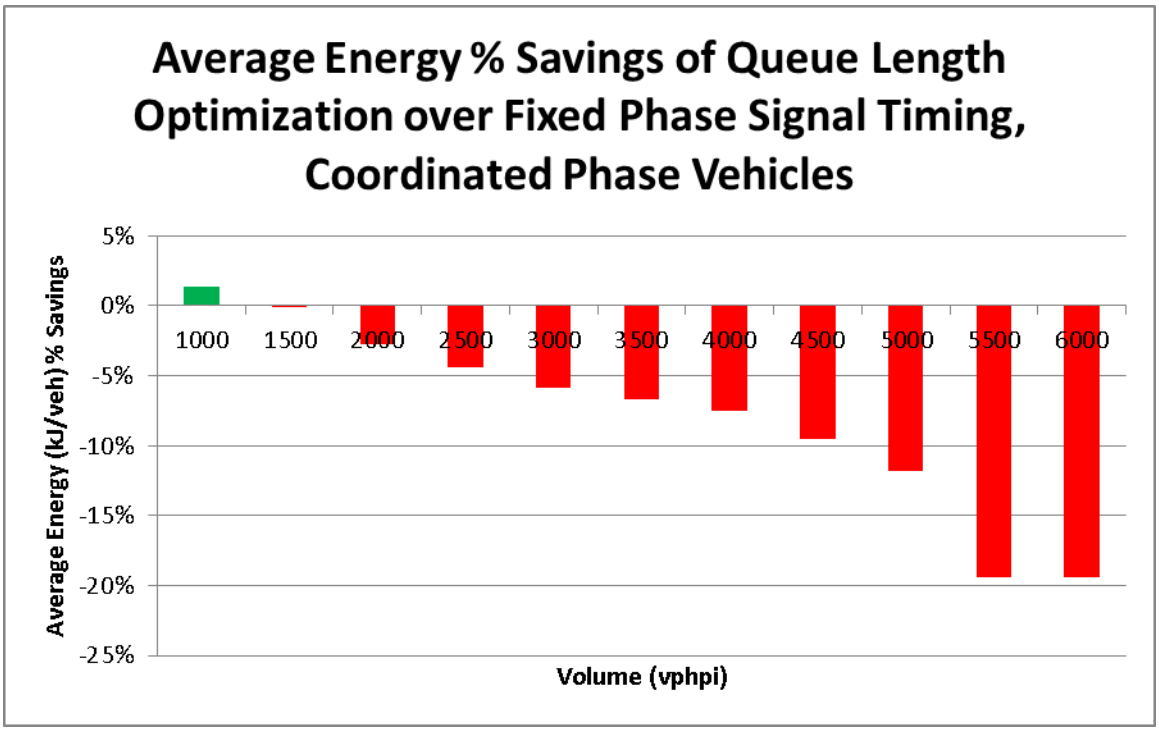

Figure 109: Coordinated Phase Vehicle Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

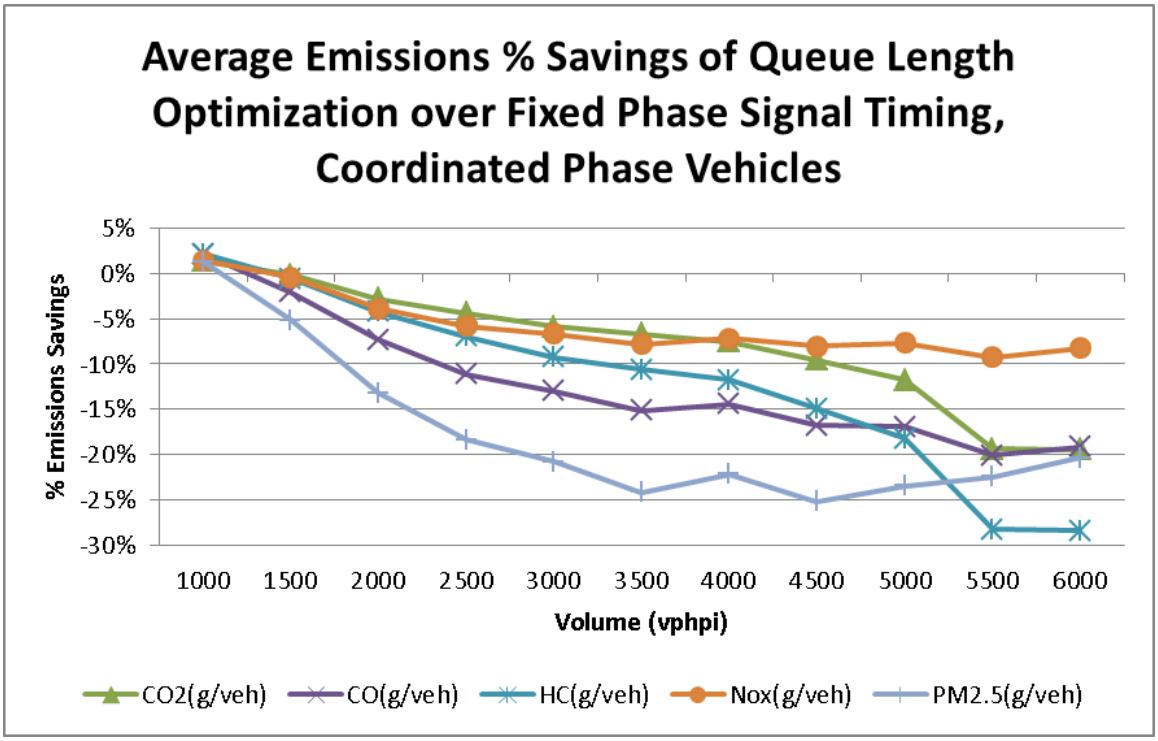

Figure 110: Coordinated Phase Vehicle Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

Table 45: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results for Coordinated Phase Vehicles

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	7472.2127	537.0262	10.1911	0.3290	1.2683	0.0685	130.3333
1500	7519.0645	540.3933	10.1956	0.3317	1.2741	0.0676	132.6929
2000	7486.7661	538.0720	9.9620	0.3287	1.2613	0.0650	134.3591
2500	7501.3055	539.1170	9.9366	0.3299	1.2612	0.0642	136.0377
3000	7614.5635	547.2568	10.0691	0.3351	1.2817	0.0649	138.3284
3500	7602.9540	546.4224	9.9911	0.3345	1.2758	0.0638	139.1749
4000	7695.3604	553.0636	10.1833	0.3403	1.2925	0.0654	141.6583
4500	7797.6319	560.4138	10.2841	0.3458	1.3048	0.0657	145.6000
5000	8183.8250	588.1692	10.7525	0.3683	1.3498	0.0684	162.8396
5500	8421.4687	605.2486	11.0373	0.3822	1.3790	0.0698	173.7539
6000	8739.0077	628.0699	11.3384	0.3985	1.4149	0.0715	187.3667

Table 46: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results for Coordinated Phase Vehicles

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	7368.2306	529.5532	9.9716	0.3219	1.2498	0.0676	127.4682
1500	7526.7237	540.9440	10.4084	0.3336	1.2796	0.0711	131.2429
2000	7696.1421	553.1200	10.6914	0.3426	1.3102	0.0736	134.9195
2500	7830.5991	562.7833	11.0419	0.3528	1.3343	0.0760	138.4751
3000	8063.1875	579.4993	11.3743	0.3660	1.3672	0.0784	146.7799
3500	8109.6911	582.8415	11.5036	0.3699	1.3758	0.0792	147.4656
4000	8275.0532	594.7260	11.6509	0.3803	1.3849	0.0799	158.2913
4500	8540.1892	613.7811	12.0111	0.3973	1.4095	0.0822	171.9603
5000	9147.3751	657.4192	12.5736	0.4353	1.4530	0.0844	210.3675
5500	10052.5094	722.4705	13.2548	0.4904	1.5061	0.0855	272.3665
6000	10437.3531	750.1290	13.5144	0.5118	1.5318	0.0861	298.2313

Table 47: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results, for Coordinated Phase Vehicles

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	1.39%	1.39%	2.15%	2.18%	1.46%	1.30%	2.20%
1500	-0.10%	-0.10%	-2.09%	-0.55%	-0.43%	-5.12%	1.09%
2000	-2.80%	-2.80%	-7.32%	-4.21%	-3.88%	-13.20%	-0.42%
2500	-4.39%	-4.39%	-11.12%	-6.97%	-5.79%	-18.34%	-1.79%
3000	-5.89%	-5.89%	-12.96%	-9.23%	-6.67%	-20.80%	-6.11%
3500	-6.67%	-6.67%	-15.14%	-10.59%	-7.84%	-24.18%	-5.96%
4000	-7.53%	-7.53%	-14.41%	-11.75%	-7.15%	-22.17%	-11.74%
4500	-9.52%	-9.52%	-16.79%	-14.91%	-8.02%	-25.16%	-18.10%
5000	-11.77%	-11.77%	-16.94%	-18.20%	-7.64%	-23.45%	-29.19%
5500	-19.37%	-19.37%	-20.09%	-28.29%	-9.21%	-22.50%	-56.75%
6000	-19.43%	-19.43%	-19.19%	-28.43%	-8.26%	-20.36%	-59.17%

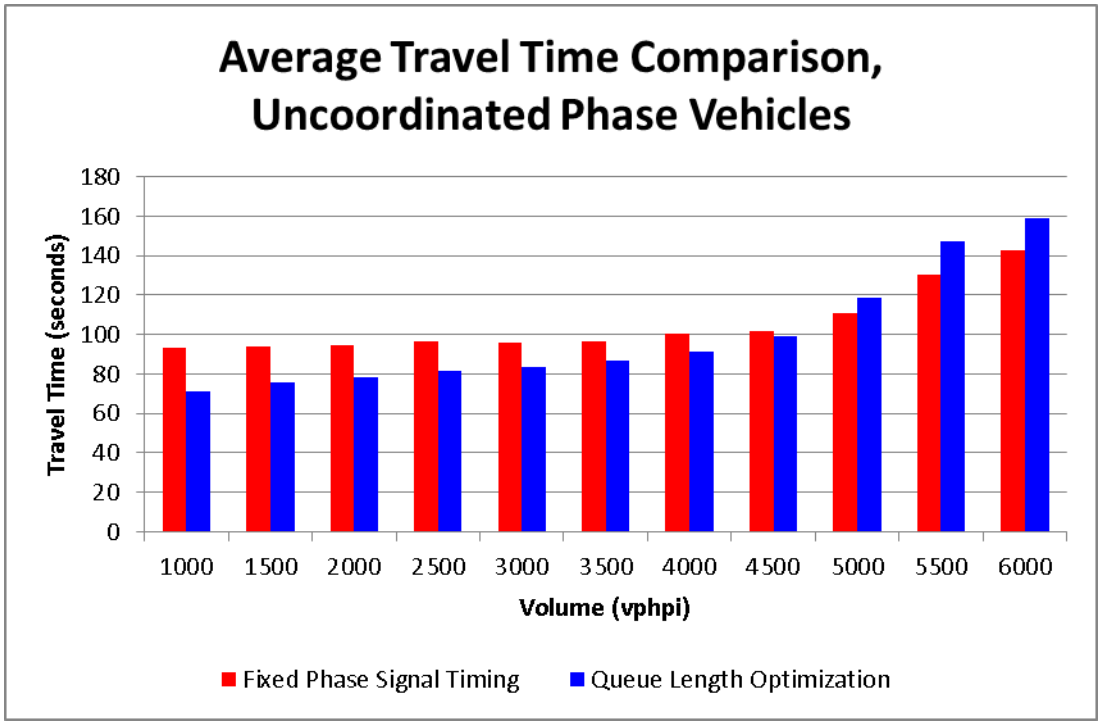

Figure 111: Uncoordinated Phase Vehicle Travel Time Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

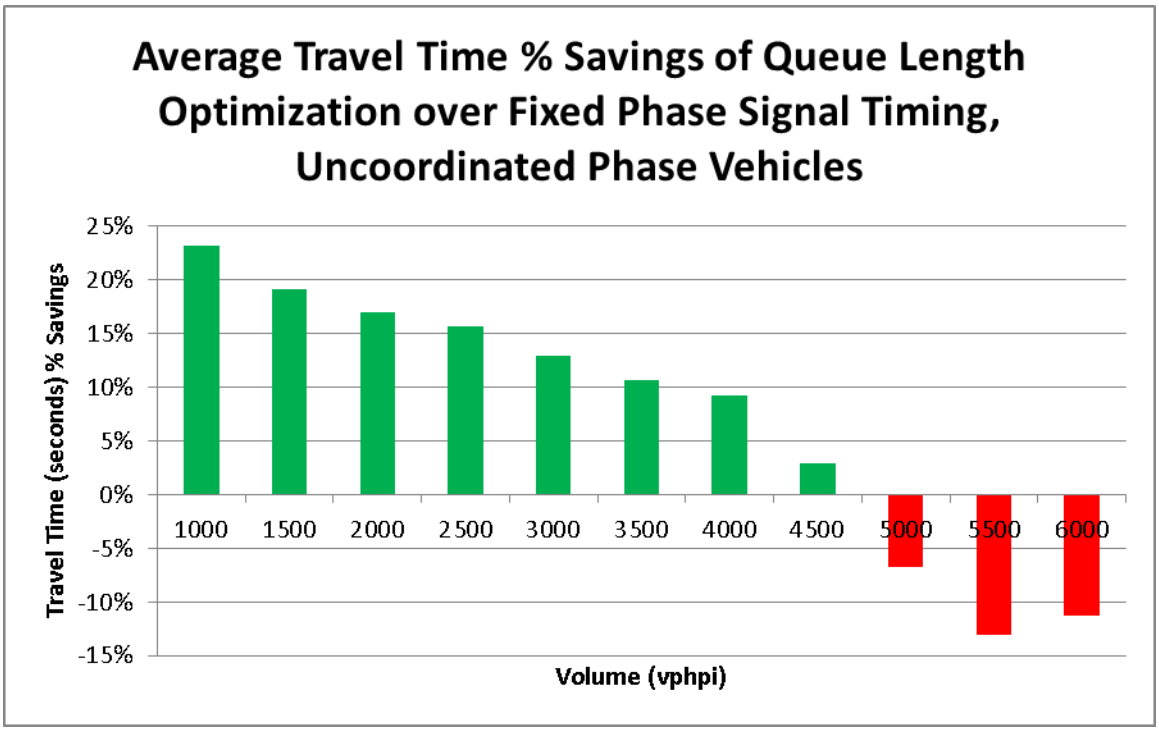

Figure 112: Uncoordinated Phase Vehicle Travel Time Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

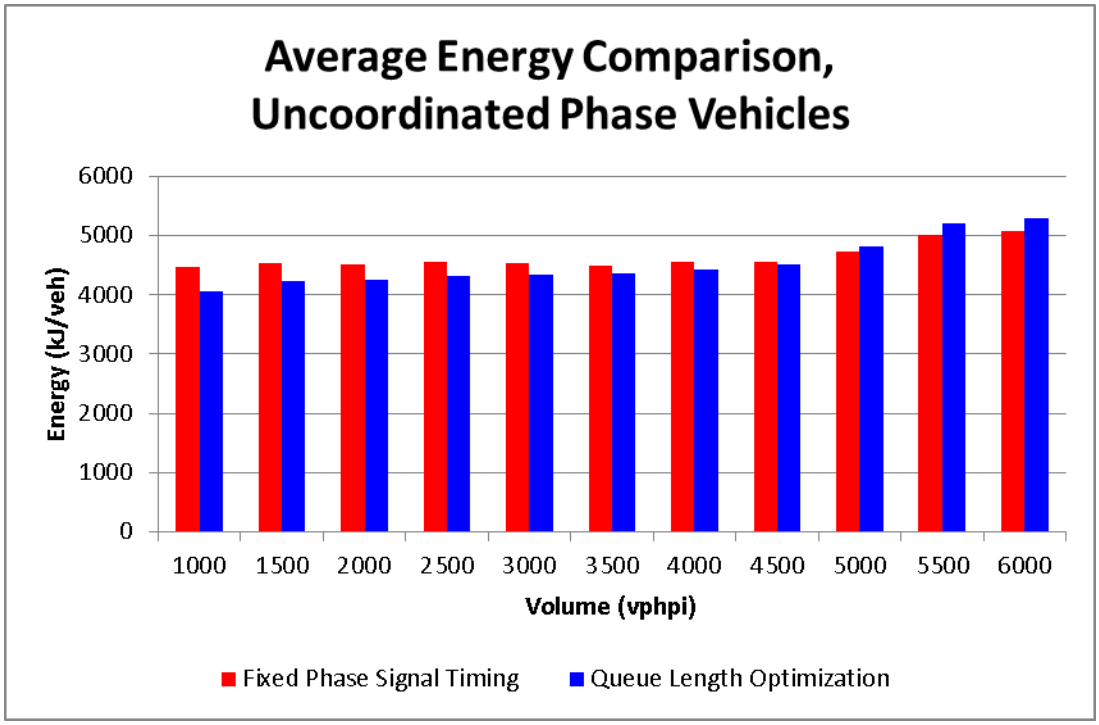

Figure 113: Uncoordinated Phase Vehicle Energy Comparison of Decentralized CV Queue Length Signal Optimization and Coordinated Fixed Phase Signal Timing for a 3-intersection Corridor

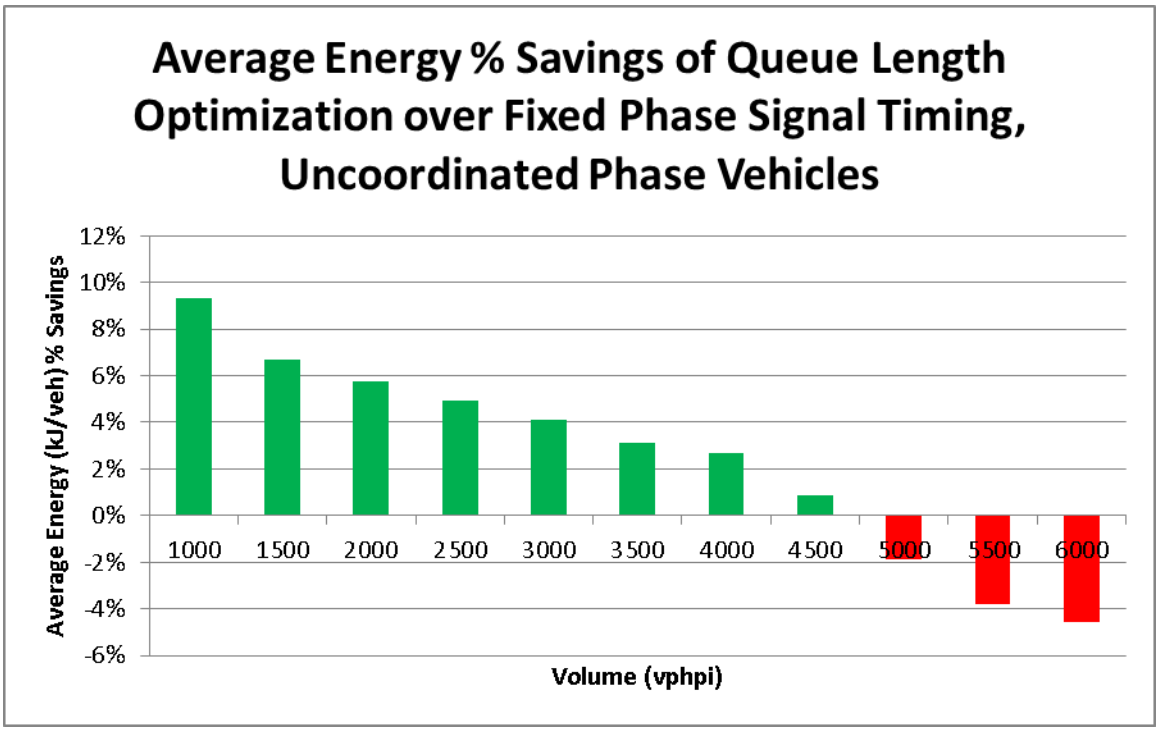

Figure 114: Uncoordinated Phase Vehicle Energy Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

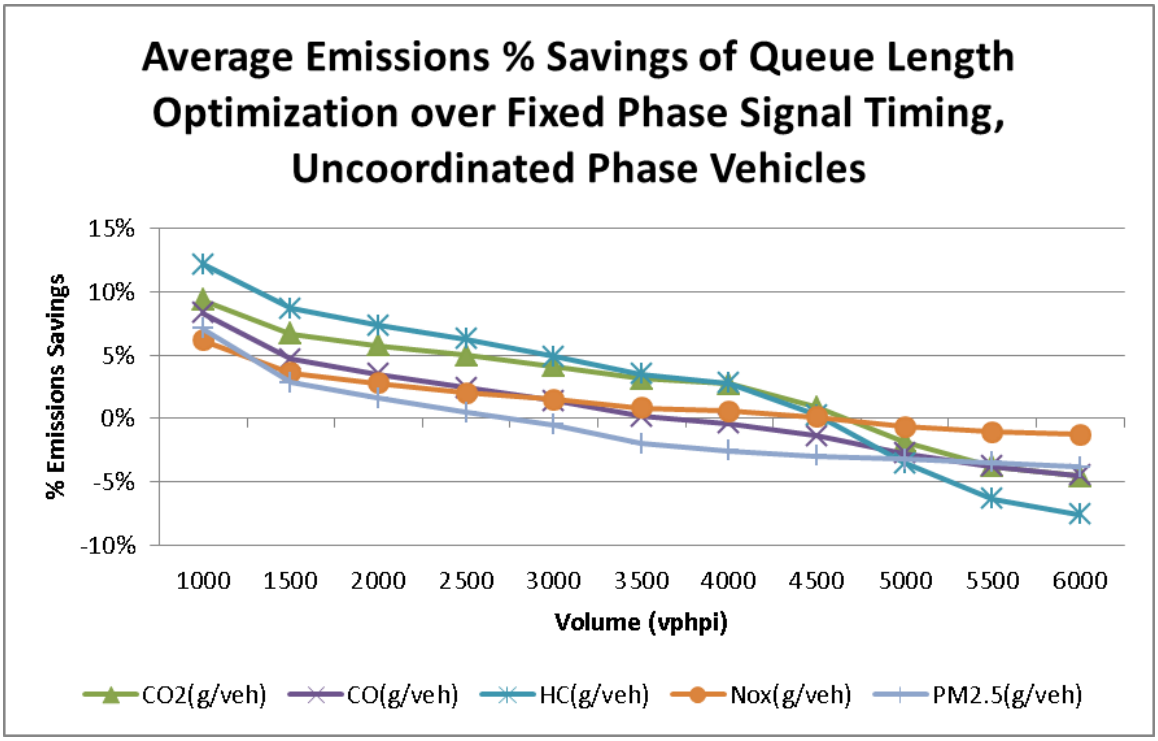

Figure 115: Uncoordinated Phase Vehicle Emissions Percent Savings of Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

Table 48: Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results for Uncoordinated Phase Vehicles

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	4472.8779	321.4644	6.8418	0.2152	0.7569	0.0496	93.0060
1500	4531.3534	325.6670	6.8907	0.2174	0.7664	0.0499	93.9077
2000	4517.9316	324.7024	6.8398	0.2168	0.7616	0.0494	94.4773
2500	4550.4730	327.0411	6.8464	0.2186	0.7630	0.0492	96.4714
3000	4527.7563	325.4085	6.7689	0.2169	0.7579	0.0484	96.0204
3500	4498.8905	323.3339	6.6737	0.2156	0.7493	0.0474	96.7397
4000	4550.4866	327.0421	6.6908	0.2185	0.7514	0.0472	100.3771
4500	4561.4876	327.8327	6.6579	0.2190	0.7494	0.0467	101.8819
5000	4734.1720	340.2435	6.7859	0.2284	0.7636	0.0469	111.0577
5500	5010.0231	360.0687	6.9320	0.2442	0.7775	0.0466	130.2493
6000	5071.7721	364.5065	6.8917	0.2480	0.7731	0.0456	142.4203

Table 49: Decentralized CV Queue Length Signal Optimization, Traffic Volume Sensitivity Analysis Results for Uncoordinated Phase Vehicles

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	4055.9481	291.5000	6.2741	0.1890	0.7102	0.0461	71.4441
1500	4228.4368	303.8967	6.5696	0.1985	0.7390	0.0485	75.9339
2000	4258.1839	306.0345	6.6025	0.2010	0.7405	0.0486	78.4212
2500	4324.9670	310.8342	6.6809	0.2049	0.7479	0.0489	81.3768
3000	4342.4276	312.0891	6.6755	0.2064	0.7466	0.0486	83.6141
3500	4357.6199	313.1809	6.6621	0.2080	0.7432	0.0483	86.4747
4000	4427.7913	318.2241	6.7171	0.2124	0.7472	0.0484	91.1646
4500	4520.8661	324.9133	6.7512	0.2184	0.7487	0.0481	98.9553
5000	4822.6587	346.6029	6.9737	0.2365	0.7687	0.0484	118.5128
5500	5200.0351	373.7247	7.1967	0.2597	0.7858	0.0482	147.2131
6000	5302.3626	381.0788	7.2019	0.2669	0.7829	0.0474	158.5766

Table 50: % Improvement of Decentralized CV Queue Length Optimization over Coordinated Fixed Phase Signal Timing, Traffic Volume Sensitivity Analysis Results, for Uncoordinated Phase Vehicles

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	9.32%	9.32%	8.30%	12.17%	6.17%	7.09%	23.18%
1500	6.68%	6.68%	4.66%	8.66%	3.57%	2.81%	19.14%
2000	5.75%	5.75%	3.47%	7.31%	2.77%	1.61%	16.99%
2500	4.96%	4.96%	2.42%	6.27%	1.97%	0.46%	15.65%
3000	4.09%	4.09%	1.38%	4.87%	1.50%	-0.51%	12.92%
3500	3.14%	3.14%	0.17%	3.51%	0.82%	-1.98%	10.61%
4000	2.70%	2.70%	-0.39%	2.76%	0.56%	-2.58%	9.18%
4500	0.89%	0.89%	-1.40%	0.25%	0.08%	-2.96%	2.87%
5000	-1.87%	-1.87%	-2.77%	-3.54%	-0.66%	-3.23%	-6.71%
5500	-3.79%	-3.79%	-3.82%	-6.35%	-1.07%	-3.48%	-13.02%
6000	-4.55%	-4.55%	-4.50%	-7.59%	-1.26%	-3.86%	-11.34%

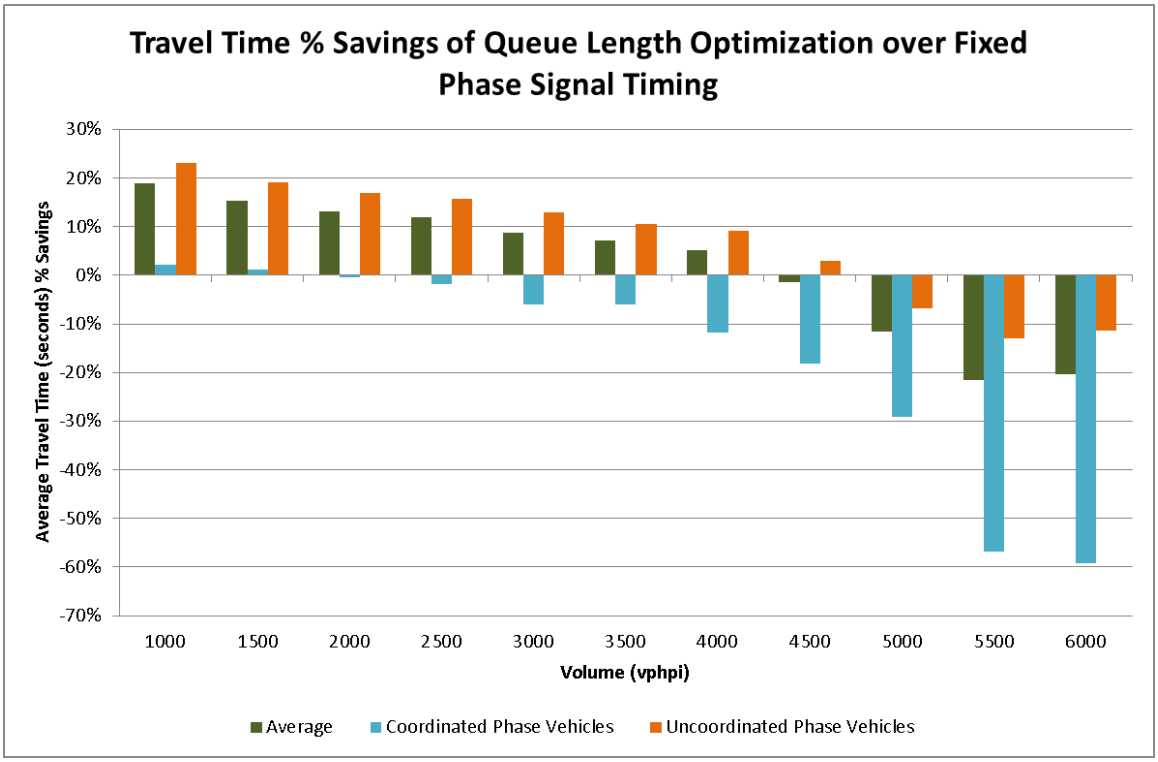

Figure 116: Comparison of Average Travel Time Percent Savings by Category for Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

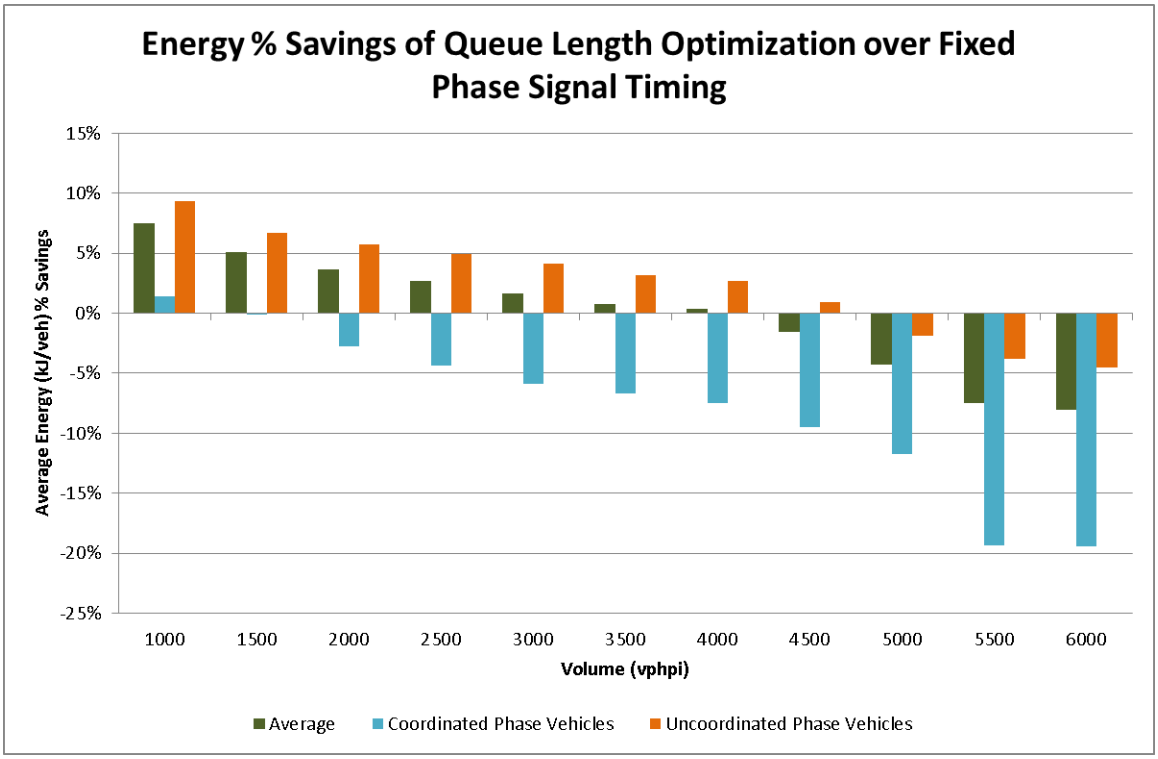

Figure 117: Comparison of Average Energy Percent Savings by Category for Decentralized CV Queue Length Signal Optimization over Coordinated Fixed Phase Signal Timing on a 3-intersection Corridor

6. Eco-Friendly Infrastructure Re-design for Intelligent Vehicles

The final major chapter in the dissertation introduces several designs for continuous flow intersections and continuous flow traffic networks starting with their derivation and continuing with a comparison of a novel continuous flow intersection with existing conventional and unconventional intersection designs. An assessment of the suitability of continuous flow networks is made in terms of network operations, travel distance, and land utilization. In addition, a benefit-cost analysis is included to quantify the potential financial impacts of deploying a continuous flow network as the basis for a new environmentally-friendly city. Finally, results are presented regarding the design parameter values for building an optimal continuous flow network.

6.1 Derivation of Continuous Flow Intersections & Networks

6.1.1 Description of the Euclidean Steiner Tree Problem and the Minimum Spanning Tree Problem

The purpose of the following section is to show the derivation of several designs for novel continuous flow intersections and continuous flow networks. A 4-leg traffic intersection may be thought of as set of roads that connect 4 points. Let the 4 points be labeled as A, B, C, and D, then a reasonable goal is to minimize the total roadway needed to connect the four points. If no additional points are permitted to be added, then the problem is considered a Minimum Spanning Tree problem. As shown in Figure 118a, a rectangle connecting the four points is a suboptimal solution. The optimal solution to the

Minimum Spanning Tree problem would be to remove either line segment AC or line segment BD. The suboptimal solution shown in Figure 118a is widely used as the basis for rectangular city blocks around the world. If additional points are permitted to be added, then the problem is known as the Euclidean Steiner Tree problem of 4 points. As shown in Figure 118b, an "X" connecting the 4 points is a suboptimal solution to the problem. Instead, the optimal solution may be shown to consist of adding 2 points instead of 1, as shown in Figure 118c. Points S1 and S2 are referred to as Steiner points to distinguish them from the original points. The Euclidean Steiner Tree problem for *N* points is considered NP-hard in terms of computational complexity and heuristics are generally employed to find near-optimal solutions. Alternatively, the solution to the Euclidean Steiner Tree problem may be approximately modelled using soap bubbles [Hwang *et al*., 1992].

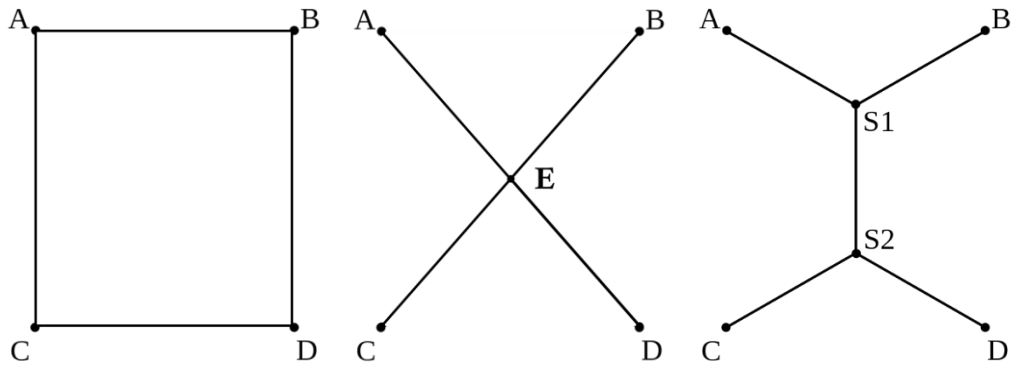

Figure 118: a) A suboptimal solution of the Minimum Spanning Tree problem, b) A suboptimal solution to the Euclidean Steiner Tree problem, c) Optimal solution to the Euclidean Steiner Tree problem; adapted from [Derksen, 2007]

The differences in the total roadway lengths of the solutions shown in Figure 118 may be illustrated as follows. Let 1, 2, 3, and 4, index the corners of the unit square, then

the total roadway lengths for the diagrams shown in Figure 119 parts a, b, and c are 3, $2\sqrt{2} \cong 2.828$, and $1+\sqrt{3} \cong 2.732$, respectively. The optimal solution to the Euclidean Steiner Tree problem of 4 points shown in part c uses roughly 3.4% less roadway than the sub-optimal solution in part b, and implies that 4-way intersections are suboptimal. In fact, the Steiner points added always result in 3-way intersections with 120 degrees between each leg. As an example, the 3-way intersections shown in Figure 119 part c at the Steiner points A and B, have 120 degree angles between the three legs.

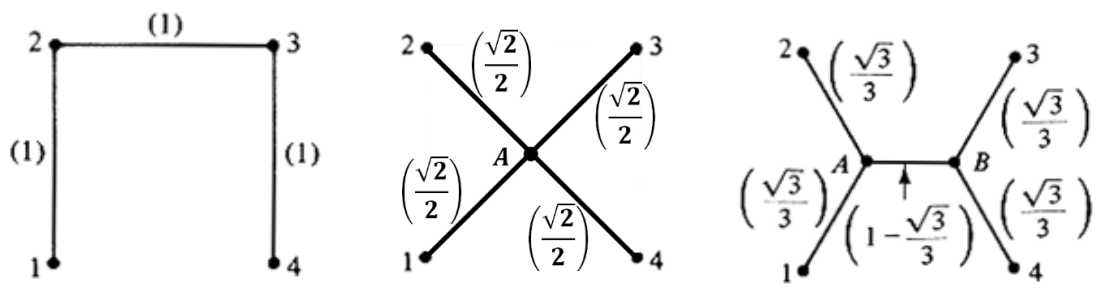

Figure 119: a) Optimal solution of the Minimum Spanning Tree problem, b) a sub-optimal solution to the Euclidean Steiner Tree problem, and c) optimal solution to the Euclidean Steiner Tree Problem for four points lying at the corners of a unit square; adapted from [Larson and Odoni, 1981]

An additional point of interest in Figure 119 part c is that every line segment is the same length, except for line segment AB. If the constraint of the unit square is relaxed, and a rectangle of length to width ratio of $2/\sqrt{3} \cong 1.1547$ is used, then every line segment becomes the same length. In the case of 4 points lying at the corners of a rectangle with a width of one unit, and a length of $2/\sqrt{3}$ units, every line segment in the optimal Steiner tree has length of $1/\sqrt{3}$ units, as shown in Figure 120 below. A noteworthy comparison may be drawn with the notion of the Steiner ratio. The Steiner

ratio is defined as the maximum possible ratio of the length of a minimum spanning tree to the length of an optimal Steiner tree for a given set of points [Ganley, 2004]. Perhaps coincidentally, the same length to width bounds ratio of 4 points placed at the corners of a rectangle that provides equal segment lengths for the optimal Steiner tree, $2/\sqrt{3}$, is also the conjectured Steiner ratio [Ivanov and Tuzhilin, 2012].

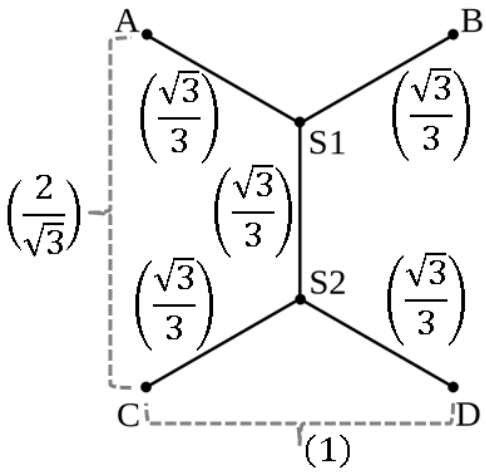

Figure 120: Steiner Tree with equivalent segment lengths

6.1.2 Tiling of the 4-Point Euclidean Steiner Tree

If the Steiner tree for 4 points shown in Figure 120 is tiled, and Steiner points are considered acceptable as network points, then the resulting tiling is a hexagonal tiling of the Euclidean plane, as shown in part in Figure 121 below. For illustrative purposes, 4 of the Steiner Trees shown in Figure 120 are arranged around a central 4-point Steiner Tree labeled A, B, C, D, S1, and S2 in Figure 121. The top left 4-point Steiner Tree is attached at A and C to the central 4-point Steiner Tree. Likewise, the top right 4-point Steiner Tree is attached at B and D to the central 4-point Steiner Tree. The bottom left 4-

point Steiner Tree is attached at S2 to the central 4-point Steiner Tree with the top right leg of the former overlapping the bottom left leg of the latter. Similarly, the bottom right 4-point Steiner Tree is also attached at S2 to the central 4-point Steiner Tree, but with the top left leg of the former overlapping the bottom right leg of the latter. The aforementioned method of tiling yields a hexagonal tiling where every network point is a Steiner point and every Steiner point is considered a network point.

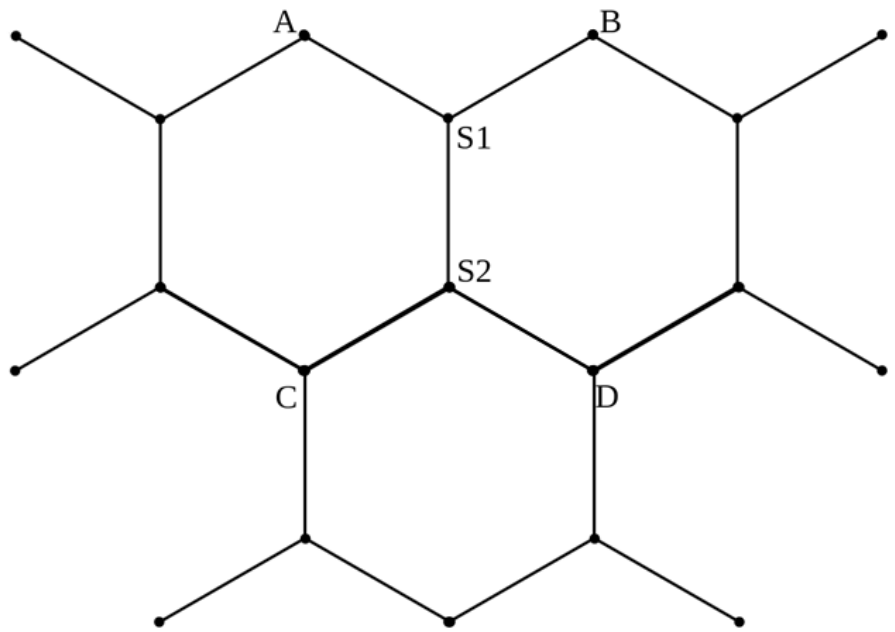

Figure 121: Tiling of an equivalent segment length Steiner Tree

6.1.3 The Hexagonal Tree as an Urban Transportation Network

6.1.3.1 Optimality of the Hexagonal Tiling

In the context of urban transportation, tilings, also referred to as tessellations, may be treated as maps of potential traffic networks, where line segments represent roadways,

vertices represent traffic intersections, and cells represent land parcels available for city blocks. The hexagonal tiling is of particular interest for urban transportation due to its minimization of total roadway needed in the traffic network. Minimizing the amount of roadway used in a transportation network also maximizes the utilization of land. According to the Honeycomb conjecture, the hexagonal tiling, or honeycomb, is the optimal method of dividing a surface into equal cells with minimal total perimeter. The oldest existing reference to the Honeycomb conjecture is as early as 36 B.C. by Marcus Terentius Varro, but the conjecture was not proven until 1999 by Thomas Hales [Hales, 2001].

6.1.3.2 Operational Modalities for Hexagonal Traffic Networks

Given that a hexagonal traffic network is optimal in terms of land utilization, numerous questions arise including how the network would operate and whether there are any benefits with respect to the existing iron grid urban traffic network. The following paragraphs will precisely describe the hexagonal traffic network's operational modalities and their mobility, safety, and environmental implications. In terms of traffic operations, there are several different methods of traffic control that may be applied to a hexagonal traffic network. As previously stated, hexagonal traffic networks are entirely composed of road segments meeting at 3-way intersections. The 3-way intersections may be either signalized or unsignalized, and permit all movements or restrict some movements. A summary of various control methods for 3-way intersections is shown in Table 51 below.

Table 51: Operational Modalities for 3-way Intersections

	All Movements Permitted	Some Movements Restricted
Signalized Control	3-phase signal control	---
Unsignalized Control	Stop-signs or round-about	No ROW required

6.1.3.2.1 3-Phase Signal Control of a Hexagonal Traffic Network

There are a total of 6 movements available for 3-way intersections, a left and a right turn from each of the three legs. If all of the 6 movements are permitted, then the hexagon network may be operated by using stop-signs, a round-about, or 3-phase signal control. The signal phase diagram for 3-phase signal control of a 3-way intersection is shown in Figure 122 below. The order of the 3 phases may be modified to provide coordinated progression for some paths through the hexagonal network. The cycle length may be set to provide a sufficiently large green band for signal coordination.

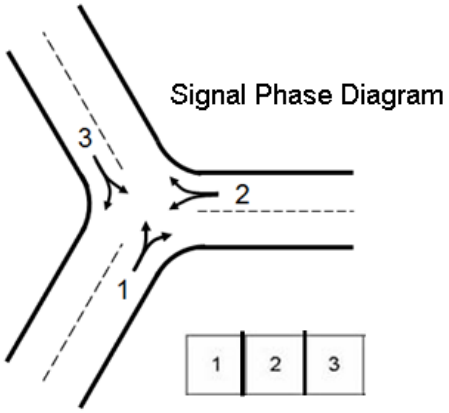

Figure 122: Signal Phase Diagram for a 3-way Intersection

One method of signalizing a hexagonal network is to treat each hexagon as a cell. The cell is defined as including only the lanes which permit clock-wise movement around the hexagon; (this definition is for right-hand drive networks). If each roadway segment has the same number of lanes, then the dividing line between hexagonal cells is the middle of the road. When the cell is "active," all movements circulating clock-wise around the hexagon are permitted, and all left-turn movements out of the hexagon are permitted. These movements are illustrated in Figure 123 below. Because the left-turns coming out of the hexagonal cell are given a green light, none of the immediately adjacent hexagons are permitted to be active. However, if the length of a hexagonal cell is defined as one unit, then all hexagonal cells exactly one unit away from an active cell may also be active. The entire network functions using 3 "network" phases, where the phases determine which hexagonal cells are active. Only one-third of the total hexagonal cells are active at any given time. Given a hexagonal network with traffic movements as described in Figure 123, and let "network" phase 1 be represented by the color red, "network" phase 2 by the color blue, and "network" phase 3 by the color yellow, then the "network" phases may be illustrated as shown in Figure 124, parts a through c, respectively. A composite image showing all 3 phases on one image is shown in Figure 125. Note that the order of the phases may be changed at any time.

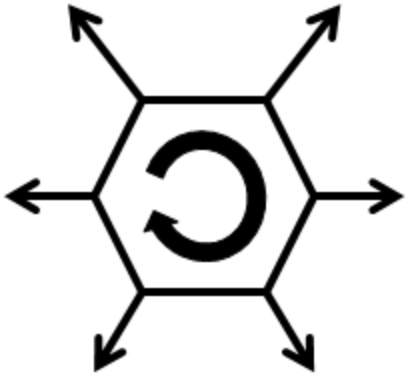

Figure 123: Traffic Movements for a Hexagonal Cell

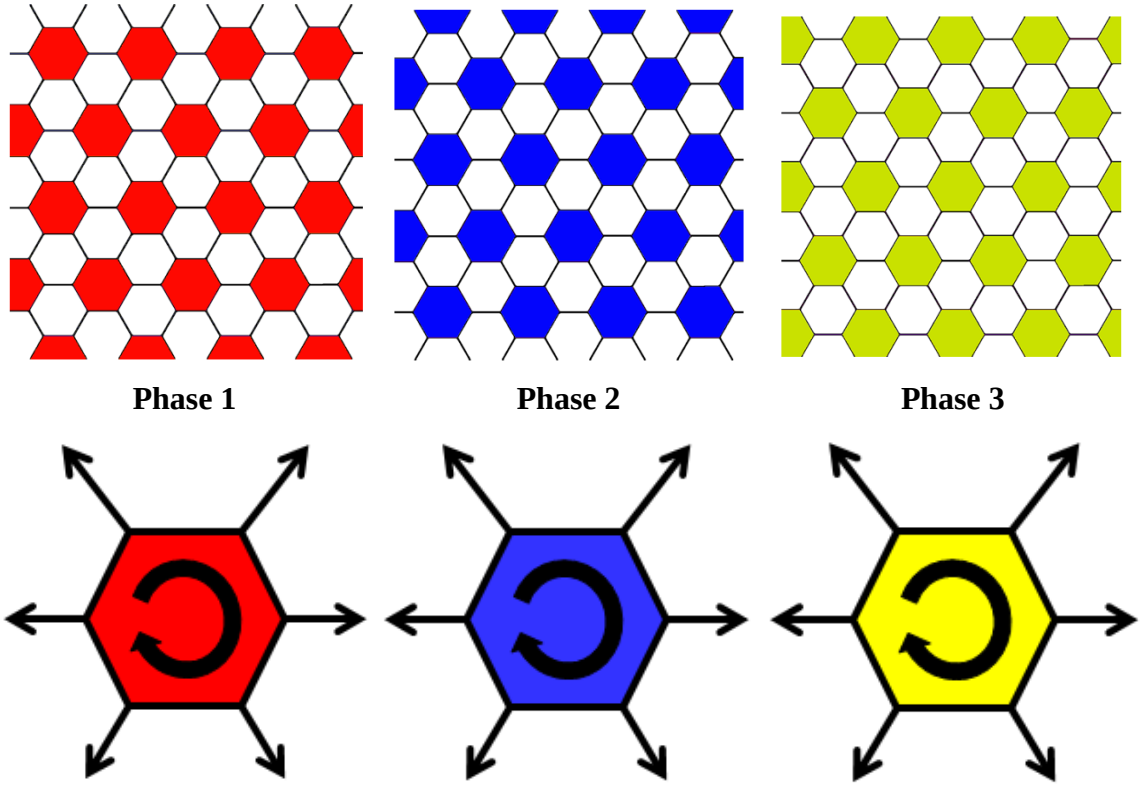

Figure 124: "Network" Phases for 3-phase Signalized Hexagonal Network, a) Phase 1, b) Phase 2, c) Phase 3

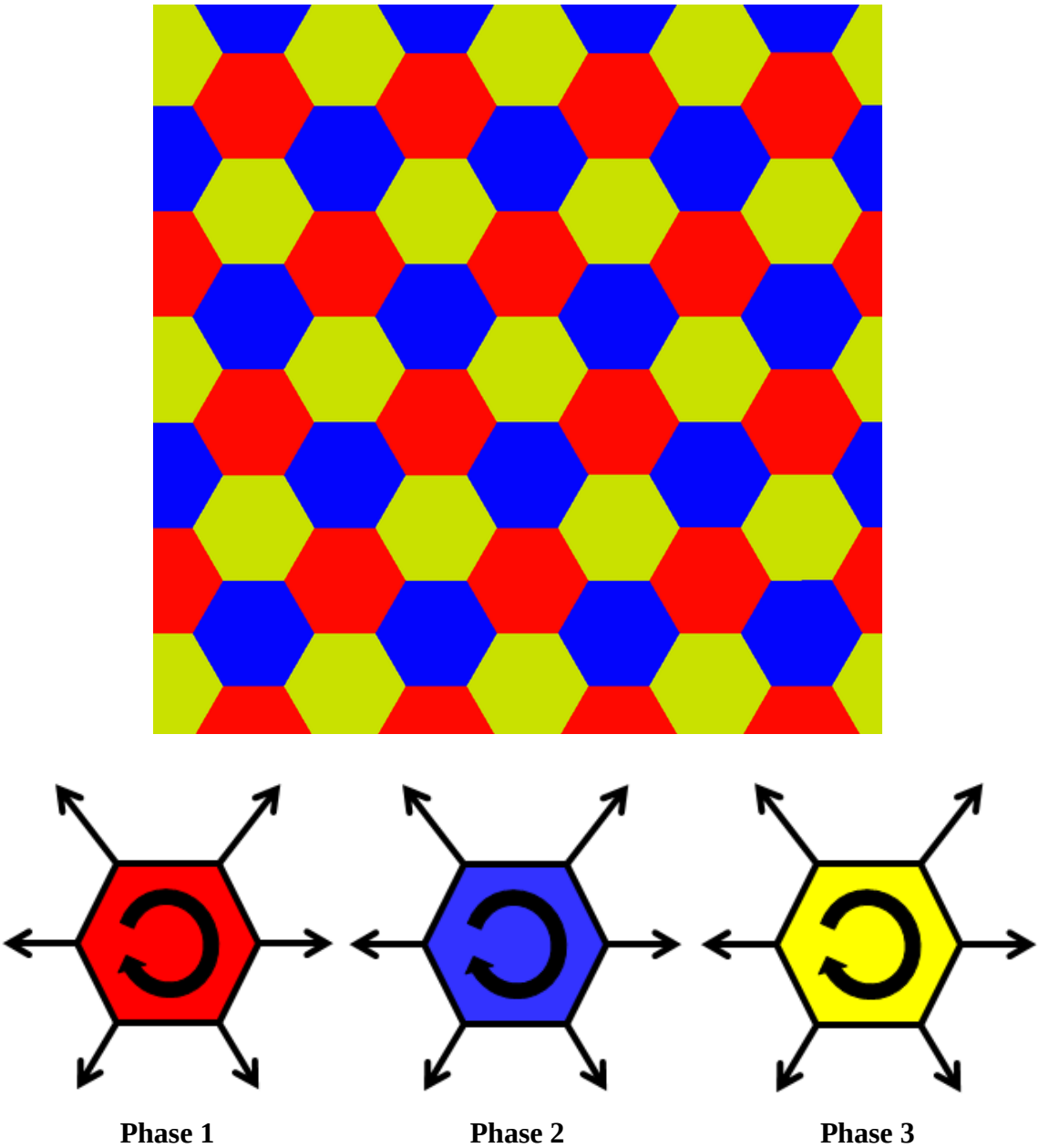

Figure 125: Composite View of "Network" Phases for 3-phase Signalized Hexagonal Network, a) Phase 1, b) Phase 2, c) Phase 3

In terms of safety, a 3-way intersection with all movements permitted has a total of 9 conflict points, as shown in Figure 126 below. Only 3 out of 9 of the conflict points are crossing points. In contrast, a conventional 4-leg intersection has 16 crossing points

out of a total of 32 conflict points. If the 3-way intersection uses a round-about, then the total number of conflict points is 6, with 0 crossing points.

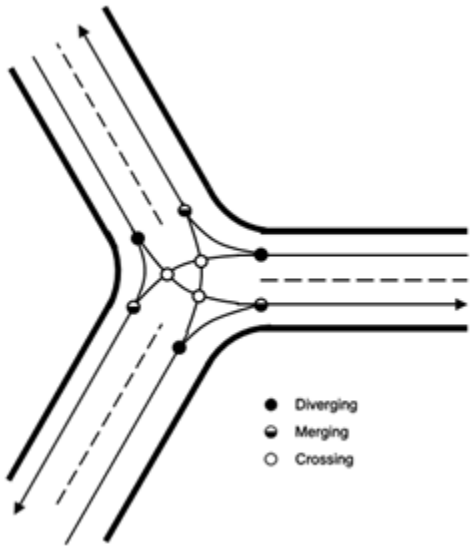

Figure 126: Conflict Point Diagram for a Three-Leg Intersection

6.1.3.2.2 Signal-less Control of a Hexagonal Traffic Network

An alternative method of operating a hexagonal traffic network is to restrict some of the movements at the 3-way intersections and convert some of the roadway links into unidirectional links. An example of a 3-way intersection operating under these principles is shown in Figure 127 below. The intersection in Figure 127 can be viewed as showing the confluence of 3 hexagonal cells. Two of the hexagonal cells circulate traffic in a clock-wise manner, and the third circulates traffic in a counter clock-wise direction. The network is shown in its entirety in Figure 128. Traffic flow is counter clock-wise around the blue hexagons and clock-wise around the white hexagons. In addition, all 6 sides of the blue hexagons represent unidirectional links. The black line-segments connecting the

blue hexagons are bi-directional links. As shown in Figure 127, left-turns are restricted at the individual 3-way intersections. However, left-turns are accommodated through the network by traveling around the blue hexagons. Significantly, the entire network operates without the use of any stop-signs, yield signs, or traffic lights, and the flow of traffic is continuous in every direction.

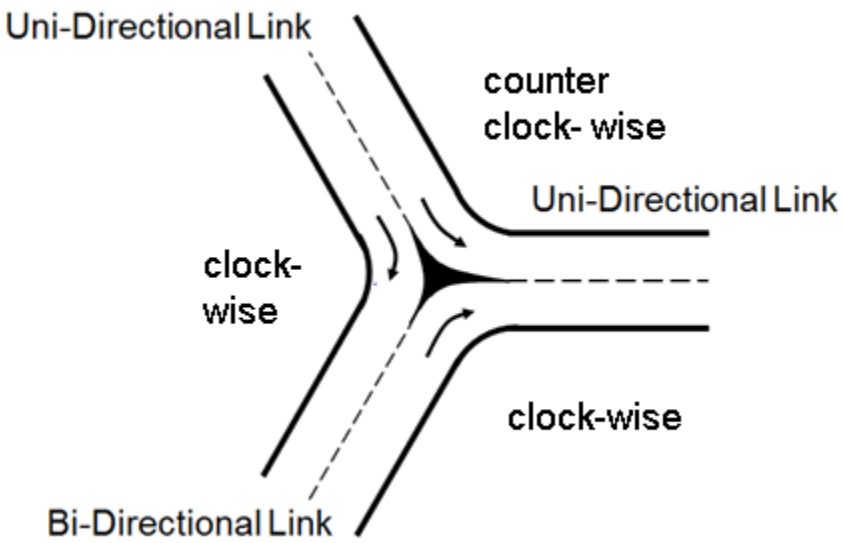

Figure 127: Example of a Restricted Left-Turn Continuous Flow 3-way Intersection

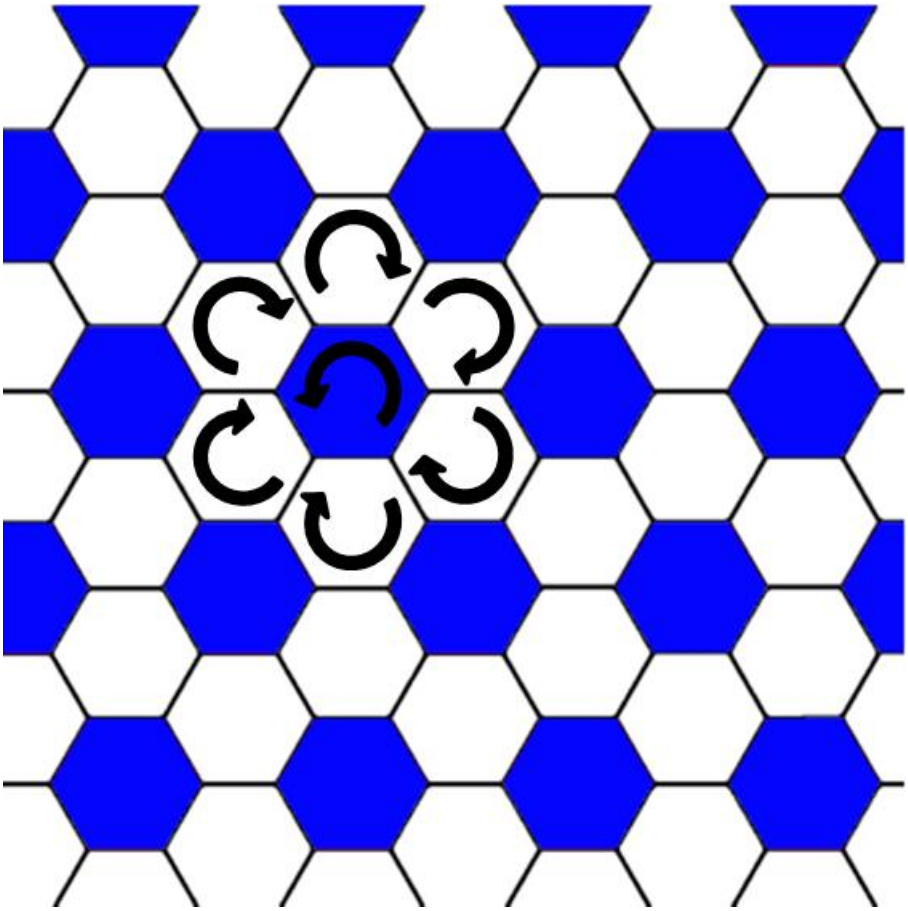

Figure 128: Network Diagram for Signal-less Control of a Hexagonal Network

6.1.4 Introduction of Signal-less 3-way Intersections as a Basis for Continuous Flow Networks

The signal-less hexagonal network shown in Figure 128 constitutes what may be termed a "continuous flow" network. The flow of traffic throughout the network is uninterrupted due to the restriction of left-turns at the three-way intersections. The use of the term "continuous" should not be confused with the prior use of the word in the patent application for a "continuous flow intersection" by Mier and Romo in 1991. In the case of the CFI UAID, the word was used to indicate that there was always at least one partial

movement in motion. In contrast, the phrase “continuous flow” is used in this dissertation to indicate that every traffic movement is constantly in motion.

6.1.4.1 Introduction of Known 3-vertex Euclidean Tilings

The restricted left-turn continuous flow 3-way intersection shown in Figure 127 need not imply that the adjoining cells are hexagons. In the case of the hexagonal tiling the angle between each of the three legs is 120 degrees. However, other angles are present in different tilings. There are a total of 11 tilings composed solely of regular polygons that have the same meeting of polygons at every vertex in the tiling. Out of the 11 1-uniform tilings, 3 are regular tilings, where a single polygon tessellates the Euclidean plane. The remaining 8 tilings are semiregular tilings, where two or more polygons tessellate the Euclidean plane. The 3 regular tilings, as shown in Figure 129 are 1) the triangular tiling, 2) the square tiling, and 3) the hexagonal tiling. Out of the 3 regular tilings, only the hexagonal tiling has vertices where 3 polygons meet. The 8 semiregular tilings, as shown in Figure 130, are 1) the snub hexagonal tiling, 2) the elongated triangular tiling, 3) the truncated square tiling, 4) the snub square tiling, 5) the rhombitrihexagonal tiling, 6) the trihexagonal tiling, 7) the truncated trihexagonal tiling, and 8) the truncated hexagonal tiling. Out of the 8 semiregular tilings, 3 of the tilings have vertices where 3 polygons meet: the truncated square tiling, the truncated hexagonal tiling, and the truncated trihexagonal tiling. In summary, the 4 tilings that consist solely of vertices where 3 regular polygons meet are shown in Figure 131.

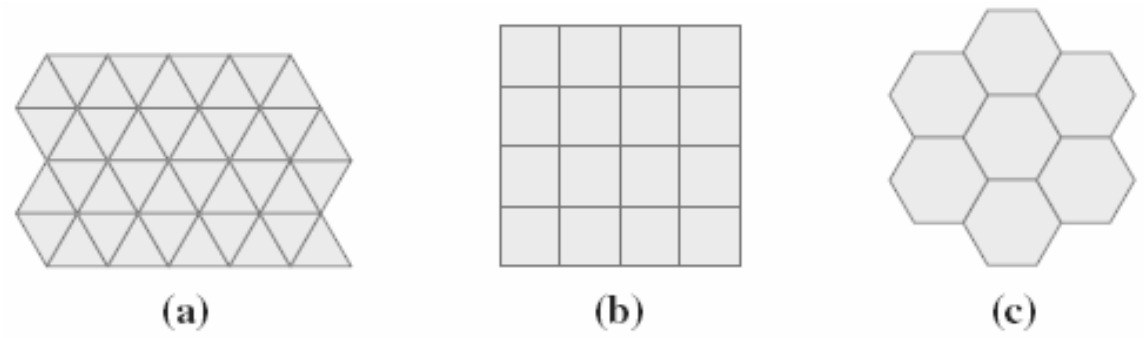

Figure 129: Regular Tessellations of the Euclidean Plane, a) triangular tiling, b) square tiling, and c) hexagonal tiling, adapted from [Parks *et al*., 2007]

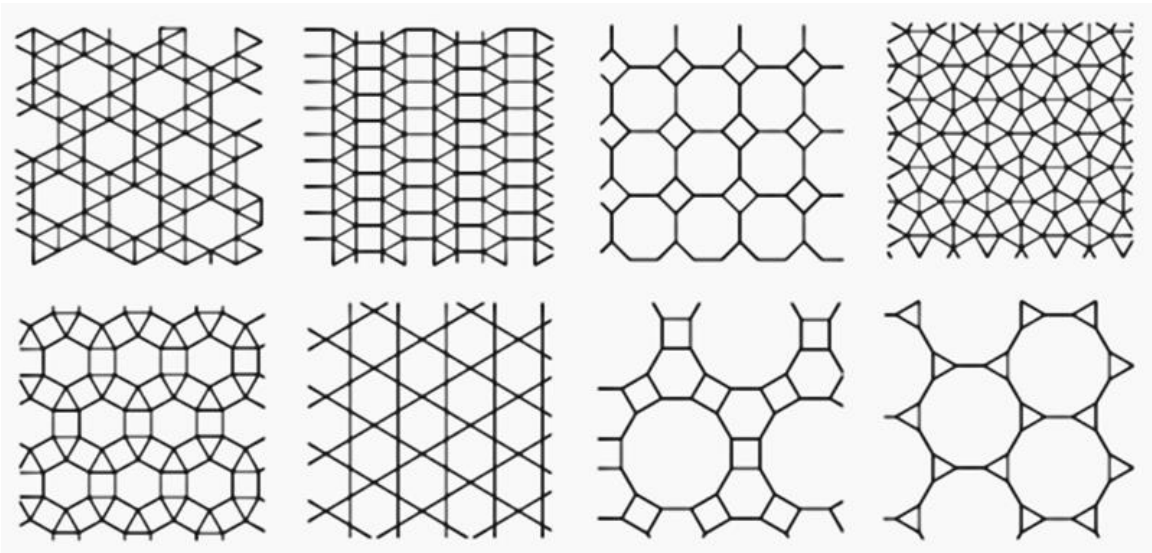

Figure 130: Semiregular Tessellations of the Euclidean Plane, from left to right, top row, a) snub hexagonal tiling, b) elongated triangular tiling, c) truncated square tiling, d) snub square tiling, bottom row, e) rhombitrihexagonal tiling, f) trihexagonal tiling, g) truncated trihexagonal tiling, and h) truncated hexagonal tiling, adapted from [Parks *et al*., 2007]

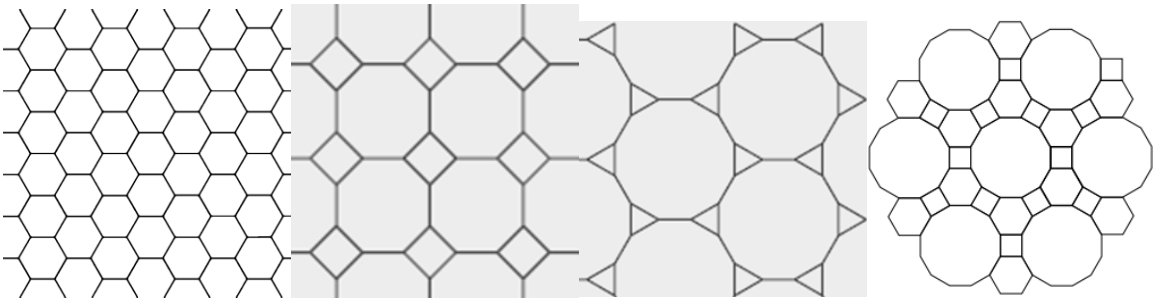

Figure 131: The 4 tilings with 3 regular polygons per vertex, a) hexagonal tiling, b) truncated square tiling, c) truncated hexagonal tiling, and d) truncated trihexagonal tiling, adapted from [Parks *et al*., 2007] and [Harris, 2000]

In the context of urban transportation, each of the tilings shown in Figure 131 above consists solely of 3-leg intersections. As mentioned previously with the hexagonal network, both signalized and unsignalized operation is possible for each of the tilings. Signalized control takes the form of a 3-phase system, where control delay is reduced in comparison to the conventional 4-phase system. Unsignalized control is based on left-turns being restricted at the three-way intersections and utilizing unidirectional links; (see Figures 127-128). In essence, restricted left-turn 3-leg intersections are a basis for continuous flow traffic networks.

Prior sections already described the operational procedures for an unsignalized hexagonal traffic network. A similar description of the operational details for unsignalized control of the remaining traffic networks shown in Figure 131 is as follows. For the truncated square tiling, the sides of the squares represent unidirectional roads, and the line segments connecting adjacent squares are bi-directional roads. Traffic flow is counter clock-wise around the squares, and clock-wise around the octagons, as shown in Figure 132. For the truncated hexagonal tiling, the sides of the triangles represent

unidirectional roads, and the line segments connecting adjacent triangles are bi-directional roads. Traffic flow is counter clock-wise around the triangles, and clock-wise around the dodecagons, as depicted in Figure 133. Lastly, there are two methods of applying unsignalized control to the truncated trihexagonal tiling. Both of the methods employ unidirectional roads throughout the network. In the first modality, traffic flow is counter clock-wise around the hexagons, and clock-wise around the dodecagons, as shown in part a of Figure 134. In the second modality, traffic flow is counter clock-wise around the dodecagons, and clock-wise around the hexagons, as shown in part b of Figure 134. Because all of the roads are unidirectional, both of the unsignalized operational modalities of the truncated trihexagonal network are equally applicable to right-hand and left-hand traffic systems without any modifications.

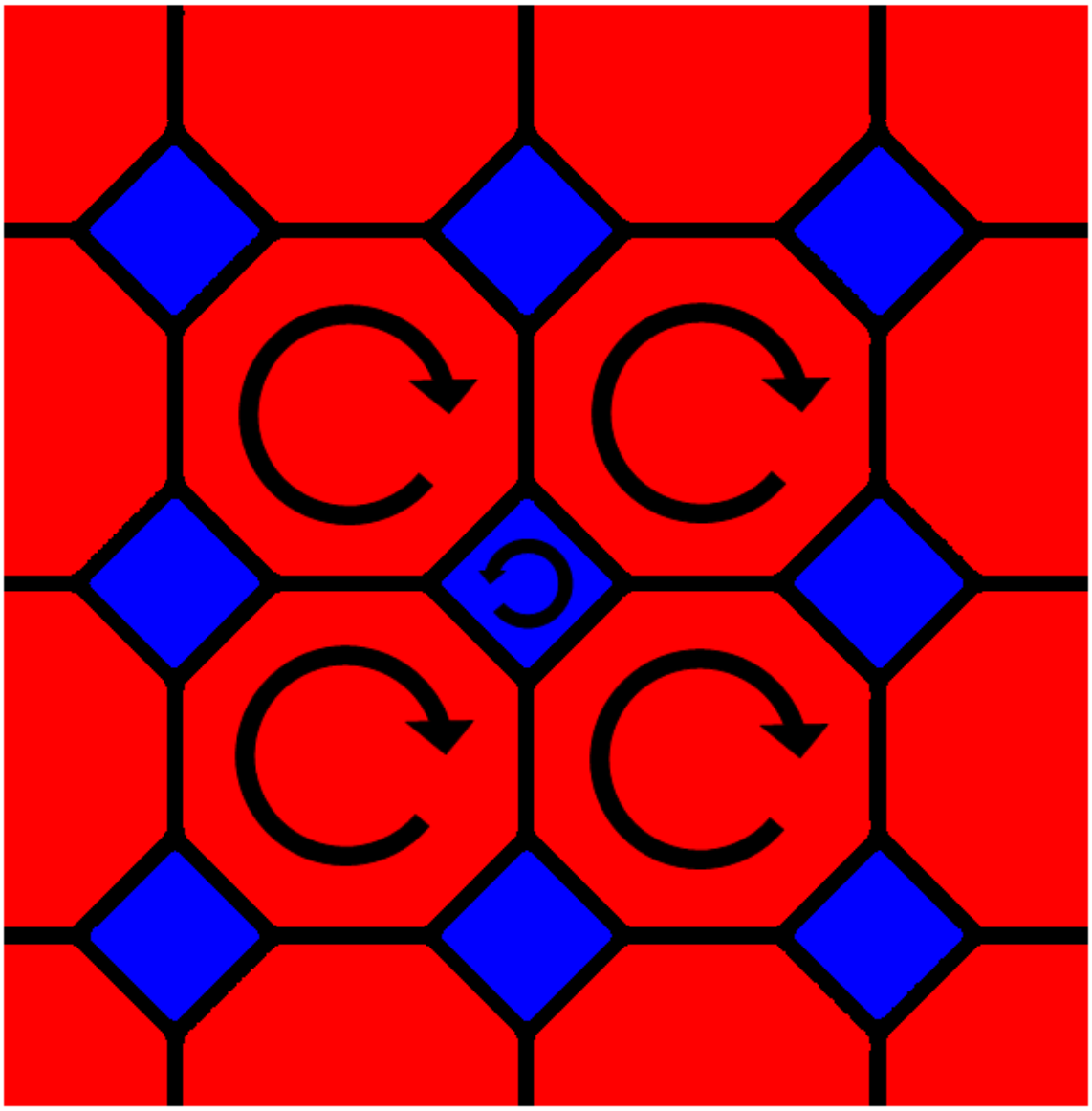

Figure 132: Network Diagram for Signal-less Control of a Truncated Square Network

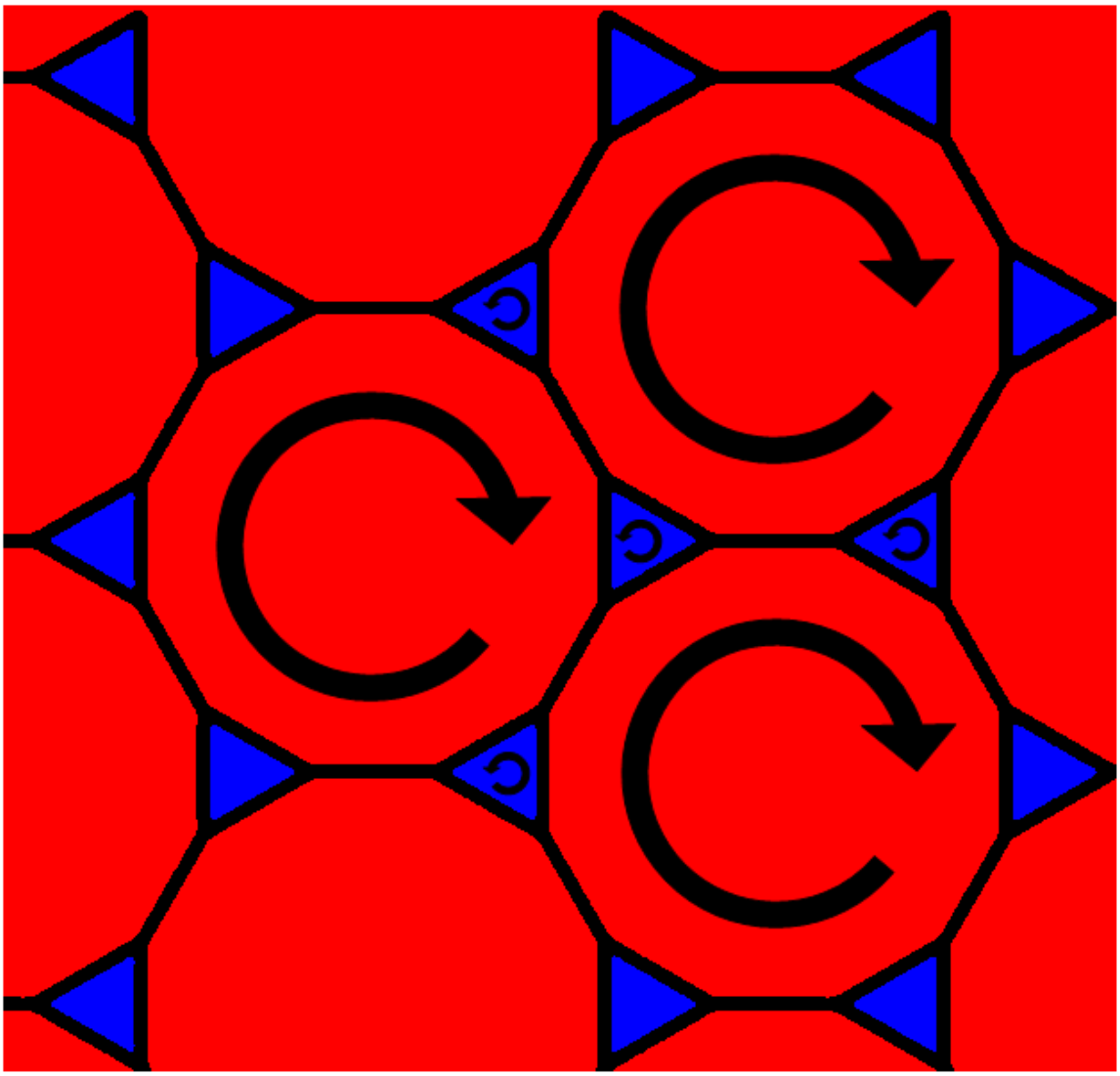

Figure 133: Network Diagram for Signal-less Control of a Truncated Hexagonal Network

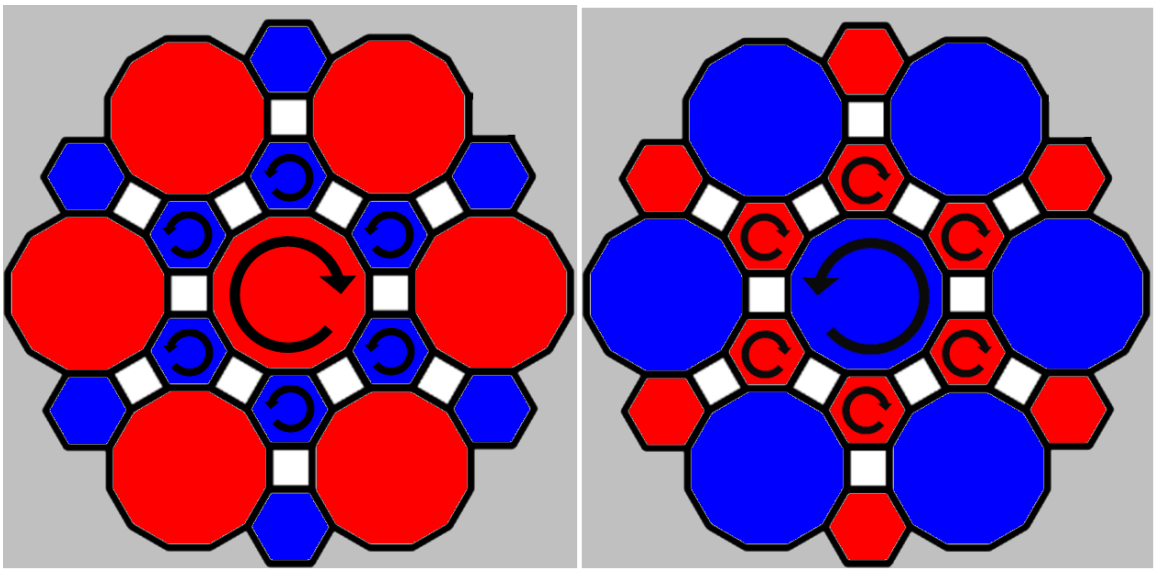

Figure 134: Network Diagrams for Signal-less Control of a Truncated Trihexagonal Network, a) Modality 1 (left), b) Modality 2 (right)

6.1.5 Additional Continuous Flow Networks

6.1.5.1 The Rounded Rectified Square Network

As shown previously in Figure 131, 4 continuous flow network designs are made possible based on applying a restricted-left turn scheme to 3-way junctions represented by 3 regular polygons meeting at a vertex in a tiling, or tessellation, of the Euclidean plane. However, additional continuous flow network designs are possible based on applying slightly different techniques. For example, instead of using restricted left-turn 3-way intersections, a modified 4-way intersection can also serve as the basis for continuous flow networks. As shown in Figure 135, traffic flows into the intersection from two opposite directions, and flows out of the intersection onto the two remaining legs. Only left-turns and right-turns are permitted in the intersection. In comparison to

the conventional 4-leg intersection, the restricted through continuous flow intersection permits only 4 movements out of the original 12 movements. While the conventional 4-leg intersection operates by providing occasional green time to a subset of the 12 possible movements, the restricted through 4-leg intersection operates by permitting continuous flow for 4 movements. The tiling of the restricted through continuous flow 4-way intersection shown in Figure 135 is achieved by having the four immediately adjacent intersections be copies of the central intersection rotated by 90 degrees. Repeating the process of adding adjacent intersections results in a square tiling, as shown previously in Figure 129. As shown in Figure 136, traffic flows in a serpentine motion through the network where every city block is essentially a square-shaped round-about. The corners of the squares are rounded to enable vehicles to maintain the speed limit during the turning motion. The network is referred to as "rectified" to indicate that the network is rotated 45 degrees with respect to the square network shown in Figure 129. A rectified square tiling is also a square tiling, simply rotated 45 degrees with respect to the unrectified square tiling. In a conventional iron-grid network, the overall flow of traffic can be represented using a straight line passing between square blocks, as shown in part a of Figure 137. In contrast, for a rectified square traffic network, the overall flow of traffic is along the diagonals of the squares, as shown in part b of Figure 137, due to movement along the diagonals of the squares being 29.2% faster than movement parallel to the squares.

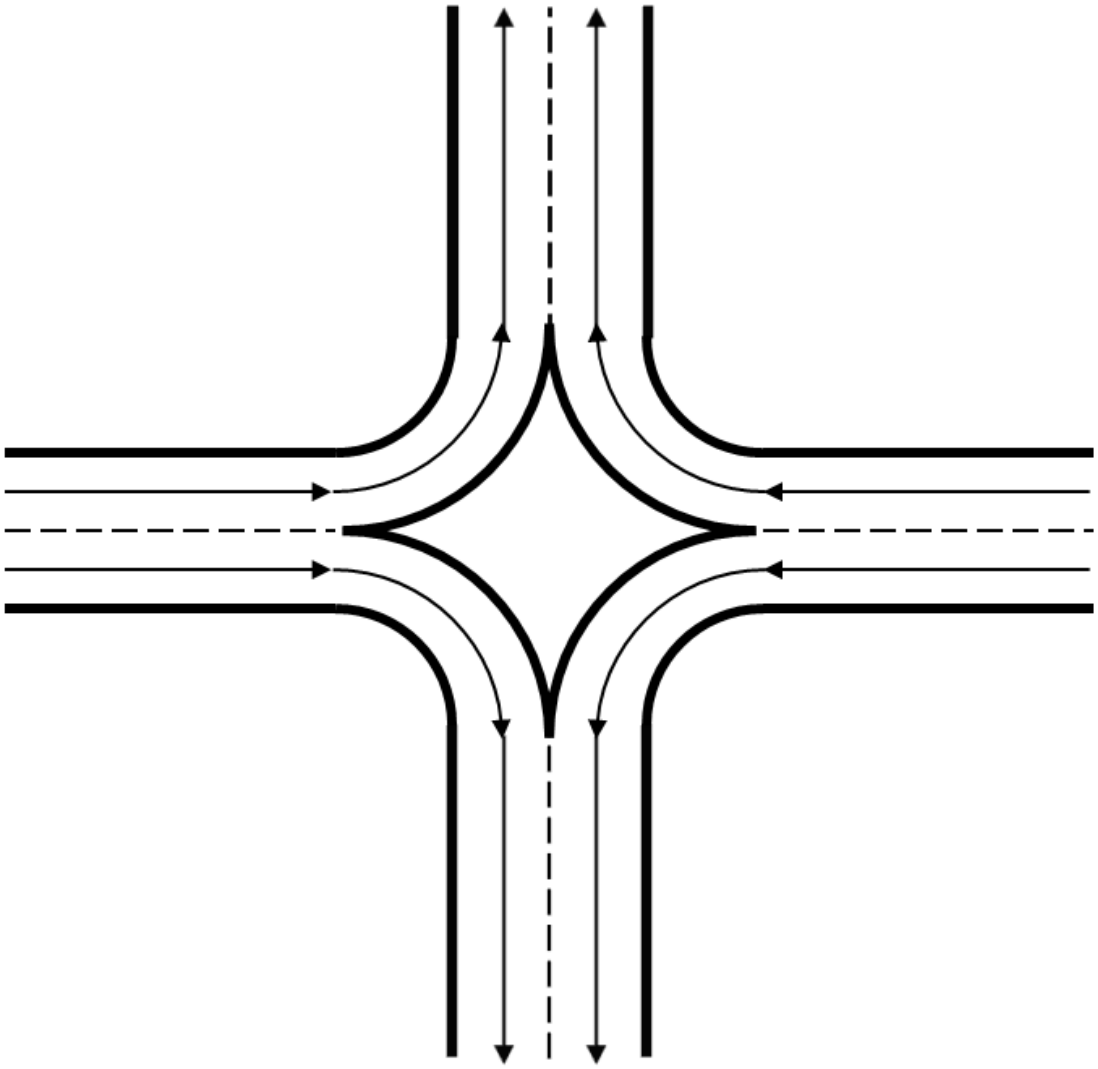

Figure 135: Example of a Restricted Through Continuous Flow 4-way Intersection

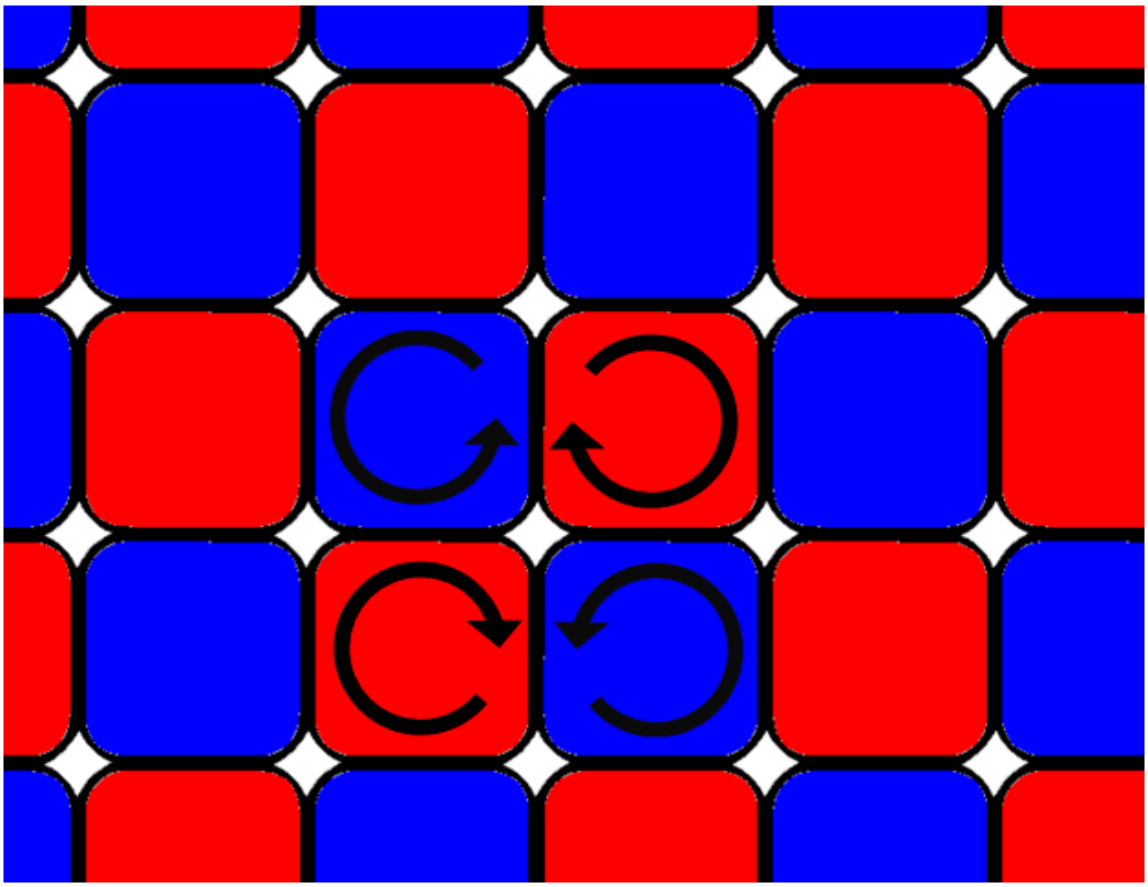

Figure 136: Network Diagram for Signal-less Control of a Rounded Rectified Square Network

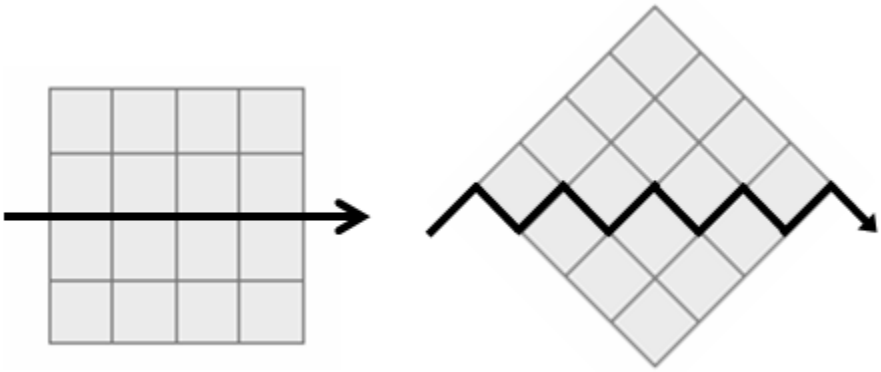

Figure 137: Comparison of Network Through Movement for a) square network (left), and b) rounded rectified square network (right)

6.1.5.2 The Martini Lattice Network

Instead of using only regular polygons, irregular polygons can also serve as a basis for continuous flow networks. For example, the martini lattice is comprised of equilateral triangles and irregular nonagons, as shown in Figure 138. The martini lattice may be derived from the hexagon tiling by truncating every other corner of each hexagon [Pozrikidis, 2014]. Every vertex in the martini lattice either has 2 nonagons and 1 triangle, or 3 nonagons. Therefore, in the context of urban transportation, a martini network is comprised solely of three-way intersections. A martini traffic network may operate using signalized control, or in as an unsignalized continuous flow network. If the network is signalized, every link is bi-directional. In contrast, if the network is unsignalized, the links comprising the triangle are unidirectional. For the continuous flow martini network, traffic flow is counter clock-wise around the triangles and clock-wise around the nonagons, as shown in Figure 139. In addition, the three-way intersections at the junction of 3 nonagons connect 3 bi-directional roads, whereas the three-way intersections at the junction of 2 nonagons and 1 triangle connect 1 bi-directional road and 2 unidirectional roads.

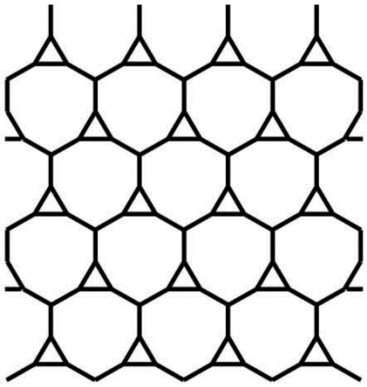

Figure 138: The Martini Lattice [Ziff and Scullard, 2006]

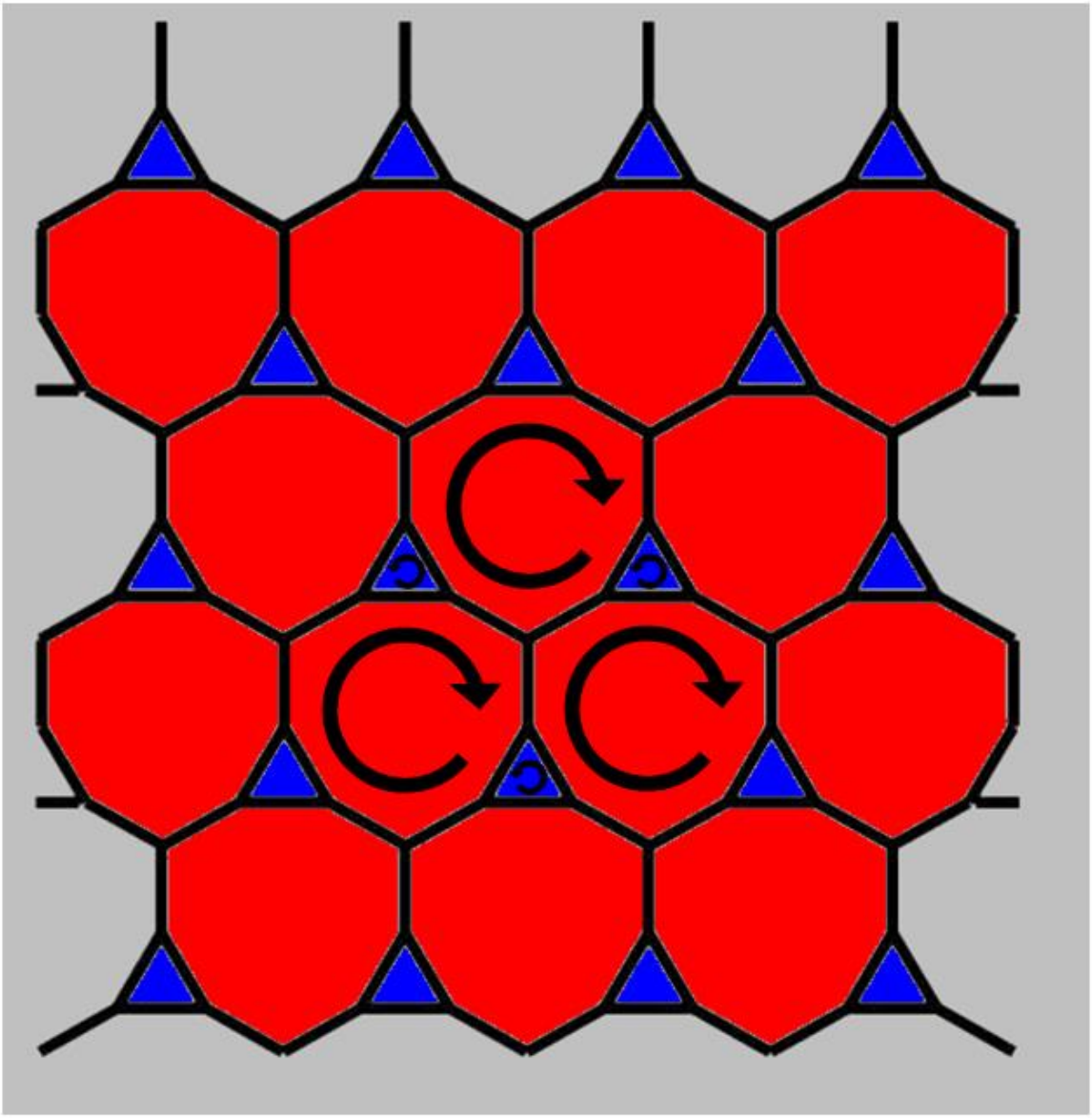

Figure 139: Network Diagram for Signal-less Control of a Martini lattice Network, adapted from [Ziff and Scullard, 2006]

The rounded rectified square network and the martini lattice network are examples of continuous flow networks not based on 3-way intersections or regular polygons. The brief list of miscellaneous continuous flow networks is illustrative of the variety of potential continuous flow networks. Many other continuous flow network

designs are possible, such as those based on 5-leg or even 6-leg intersections; however, a majority of these designs are unsuitable for urban transportation due to excessive travelling distances.

6.1.6 Network Comparisons

6.1.6.1 Network Operations Comparison

Previous sections introduced and briefly described 6 designs for potential continuous flow urban traffic networks. The list of designs included 1) the hexagonal network, 2) the truncated square network, 3) the truncated hexagon network, 4) the truncated trihexagonal network, 5) the rounded rectified square network, and 6) the Martini network. A summary of some of the operational and geometric features of the networks is provided in Table 52 below. The second column indicates whether signalized control is possible for the network design, and if signalization may be coordinated. The third column indicates whether the network can alternatively be operated as a continuous flow network without traffic signals or stop signs. The fourth column indicates whether only one-way roads are present in the unsignalized continuous flow version of the network. Finally, the last column indicates the number of sides of the polygon shaped round-abouts in each design. In addition, the direction of flow is indicated in parentheses. With the exception of the rounded rectified square network, each of the networks may operate using 3-phase signal control. In addition, each of the networks can alternatively be conFigured to facilitate continuous flow traffic. Two of the networks, the truncated trihexagonal network and the rounded rectified square network utilize only one-way

streets and have polygon round-abouts that channel traffic in either a counter clock-wise manner or a clock-wise manner. The remaining networks include bi-directional links and only have polygon round-abouts that channel counter clock-wise traffic.

Table 52: Comparison of Network Operational Modalities and Geometric Features

Network	**3-phase signalized control possible?**	**Continuous flow operation possible?**	**Unidirectional links only?** (for continuous flow modality)	**N?-sided round-about** (direction of flow around round-about: cw = clock-wise, ccw = counter clock-wise)
Hexagonal	Yes, coordinated in 3 out of 6 possible directions	Yes	No	6 (ccw)
Truncated Square	Yes, coordinated	Yes	No	4 (ccw)
Truncated Hexagon	Yes	Yes	No	3 (ccw)
Truncated Trihexagonal	Yes	Yes	Yes	6, 12 (ccw, cw / cw, ccw)
Rounded Rectified Square	No	Yes	Yes	4, 4 (ccw, cw)
Martini	Yes	Yes	No	3 (ccw)

6.1.6.2 Circuity Comparison

In the context of transportation, circuity is defined as the ratio of route distance to Euclidean distance in a traffic network [Levinson and El-Geneidy, 2009]. If a driver was able to travel from point A to point B in a straight line, then, (neglecting the curvature of the earth), the circuity would equal 1. However, due to the presence of buildings and

other obstacles, urban traffic networks have a circuity greater than 1. A conventional grid network has a circuity of about 1.25. Although the 6 continuous flow networks presented in sections 6.1.4 and 6.1.5 provide the benefit of uninterrupted movement of all traffic directions, a critical concern is the increase in Vehicle Miles Travelled (VMT), which is equivalent to the route distance. As a result, the approximate increase in route distance for each of the 6 networks relative to a conventional grid network is presented in Table 53 below. Columns 2-4 indicate the approximate increases in route distance for through movements, left-turn movements, and right-turn movements relative to a conventional grid network. The fifth column is an average of the 3 prior columns, and is an approximation of what the average increase in route distance is, summed over all OD pairs. However, in a real-world urban traffic network all OD pairs are travelled with the same frequency. Typically, a driver will make more through movements along a route, than turning movements. In order to account for realistic turning percentages, a weighted average of columns 2-4 is shown in the last column of Table 53. In terms of through movements, the truncated square network increases the route distance the least compared to other continuous flow networks. Due to the use of circuitous paths through the networks, most of the networks provide a small benefit in terms of right-turn movements, but a large penalty for left-turn movements. Although the truncated hexagon network added the least route distance for the simple average, the weighted average revealed that the truncated square network adds minimal route distance for more frequently travelled routes.

Table 53: Route Distance Comparison of Continuous Flow Networks vs. Conventional Grid Network

	Approximate % Increase in Route Distance				
Network	**Through Movement**	**Left-Turn Movement**	**Right-Turn Movement**	**Average**	**Weighted Average (5*T+ 1*L+1*R)**
Hexagonal	+33%	+67%	-33%	+22%	+28%
Truncated Square	+24%	+66%	-17%	+24%	+24%
Truncated Hexagon	+43%	-11%	-11%	+7%	+28%
Truncated Trihexagonal	+27%	+118%	+13%	+53%	+38%
Rounded Rectified Square	+41%	+41%	+41%	+41%	+41%
Martini	+215%	+373%	+8%	+199%	+208%

It is important to note that the distance a vehicle travels is not correlated with the time a vehicle takes to travel through a given traffic network. Even though a vehicle may travel a longer distance, the travel time may still be lower than a shorter route that experiences significant delay. Likewise, vehicles travelling longer distances in continuous flow networks may travel faster than vehicles in a conventional signalized grid network if the delay experienced in the conventional network is sufficiently high. The average delay in a continuous flow network can be expected to be minimal, due to the removal of interruptions to traffic flow. However, if the delay in a conventional network is low enough, then the conventional network will still outperform a continuous flow network in terms of average travel time. The point at which the travel time in a conventional grid network would match the travel time in a continuous flow network is shown in Table 54 below. The total travel time that a vehicle takes through a traffic

network is comprised of free-flow travel time and delay time. Free-flow travel time is the amount of time a vehicle would take while traveling the speed limit along a certain route without any interruptions. Delay time is the difference between total travel time and free-flow travel time. The amount of delay that a vehicle experiences can be expressed as a percentage of the total travel time. As shown in Table 54, if the percent delay experienced in a conventional network is less than 19%, then the conventional network will outperform all of the continuous flow networks listed in terms of travel time. Conversely, if the percent delay in a conventional network is greater than 68%, then any of the continuous flow networks listed will outperform the conventional network in terms of travel time. The values in Table 54 were calculated by taking the values in the final column of Table 53 and dividing them by the quantity 1 plus the value. The assumption is that the percent delay of a continuous flow network is negligible.

Table 54: Maximum Percent Delay of Conventional Network under which a Conventional Network will outperform a Continuous Flow Network in terms of Travel Time

Network	**% Delay on Conventional Network where Travel Time is equal with Continuous Flow Network**
Hexagonal	22%
Truncated Square	19%
Truncated Hexagon	22%
Truncated Trihexagonal	28%
Rounded Rectified Square	29%
Martini	68%

6.1.6.3 Land Utilization Comparison

Land utilization is the percentage of land in an urban setting that is available after roads are constructed. Specifically, land utilization equals land area divided by the sum

of land area and roadway area. For a conventional square grid network the land inside the square is referred to as the city block. For consistency, the term "cell" will be defined as including both the land area and half of the width of the surrounding roadway. A high land utilization percentage equates to selecting polygons with minimal perimeter to maximize the packing factor percentage. For urban transportation networks, the packing factor is equivalent to land utilization, and minimizing the perimeter of a polygon equates to reducing the amount of roadway in a transportation network. The ratio of the perimeter of a regular polygon to its area is minimized as the number of sides of the polygon is increased. Increasing the number of sides to infinity yields a circle. The circle provides minimal perimeter for a given area. However, the circle is not a polygon, and only a few polygons, (the triangle, square, and hexagon), can tile the Euclidean plane. Out of the three shapes which can tile the Euclidean plane, it is the hexagon that has most sides. As stated earlier the proven Honeycomb conjecture confirms that the hexagon tiling provides the minimal perimeter tiling of a single regular polygon. However, tiling involving multiple types of regular polygons can exceed the packing factor of a hexagonal tiling.

The land utilization percentage for a given tiling increases as the cell size is increased. Figure 140 and Table 55 show the case of the area of the largest polygon cell in each network being constrained to be equal to one million square feet. In the case of the hexagon, square, and rectified square networks all of the polygons are identical. However, in the truncated square network octagonal cells are the largest. Similarly, dodecagonal cells are the largest cells in both the truncated hexagonal network and the

truncated trihexagonal network. Irregular nonagons are the largest cells in the Martini lattice network. As the number of lanes is increased, the land utilization percentage drops. The ranking of the networks based on highest packing factor percentage remains the same regardless of the number of lanes. The ranking is also largely unaffected by different cell areas. The hexagon, square, and rectified square networks have the highest land utilization percentage when the area of the largest cells in each network is constrained to be equal.

Alternatively, if the cells in multiple polygon type tilings are constrained to have the same average area as cells in single polygon tilings, then the ranking of multiple polygon type networks improves. As shown in Figure 141 and Table 56, the truncated trihexagon, truncated hexagon, and truncated square networks all have a higher land utilization percentage than the hexagon network.

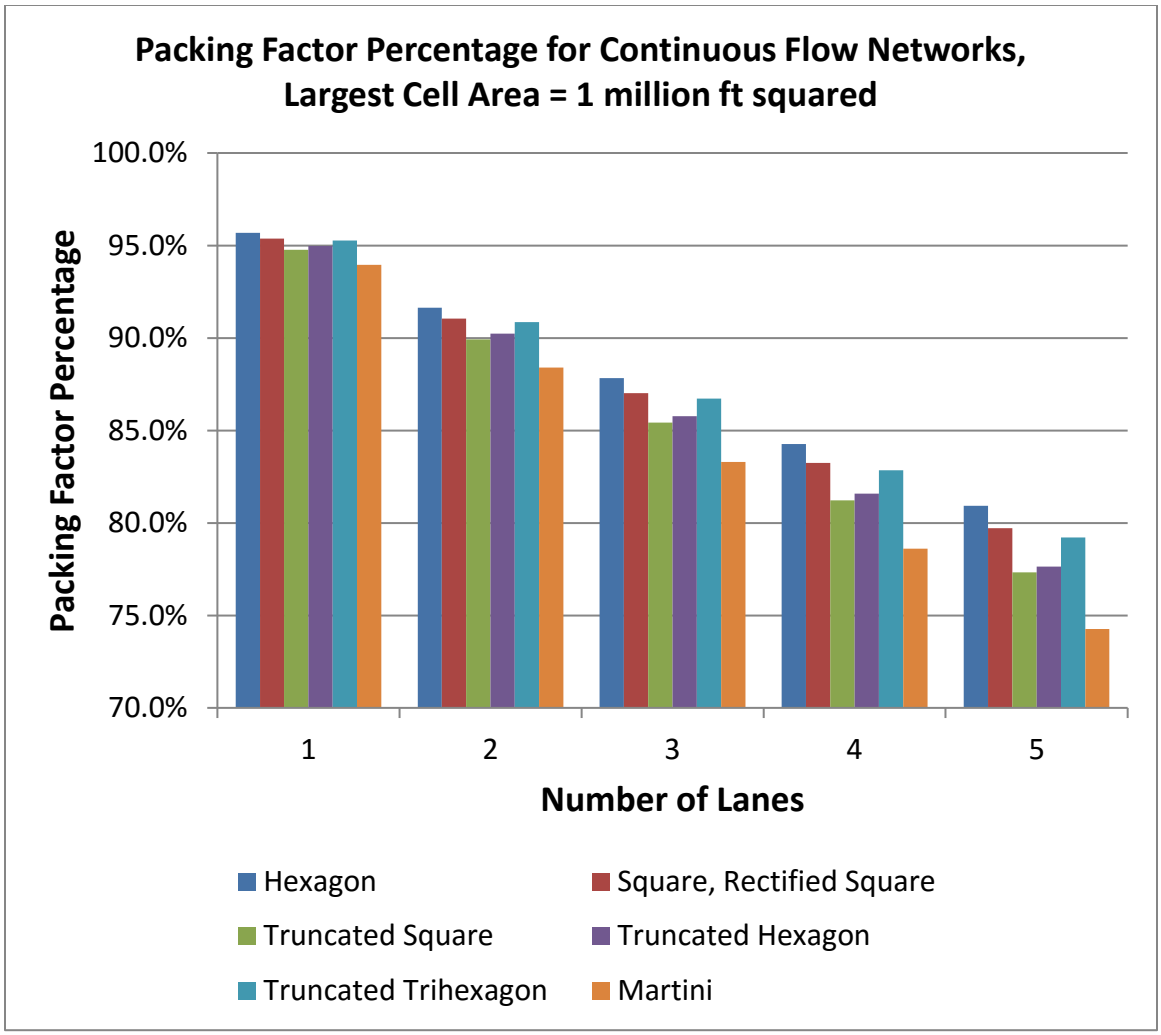

Figure 140: Packing Factor Percentage for Continuous Flow Networks with Largest Cell Area of 1 million square feet

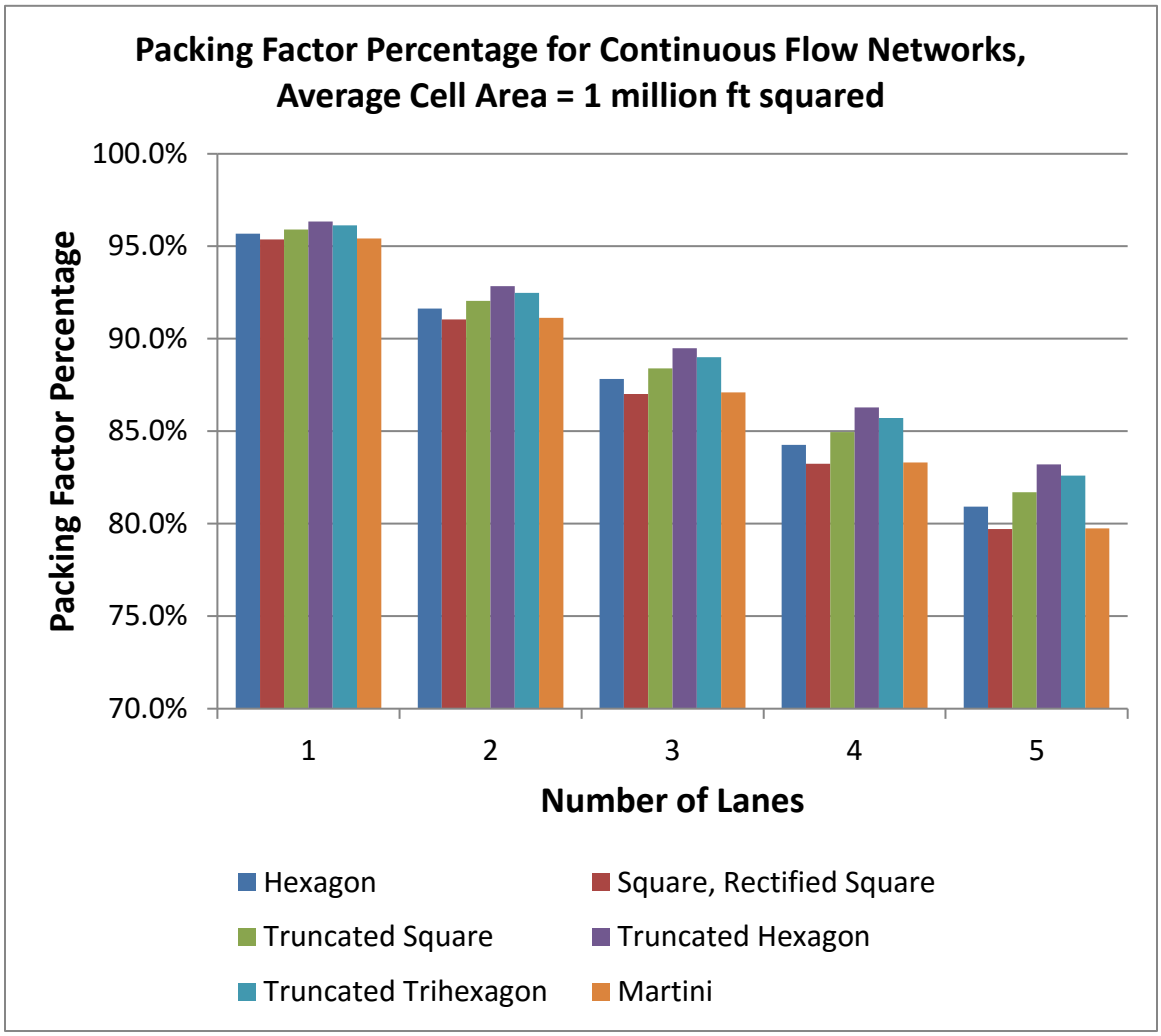

Figure 141: Packing Factor Percentage for Continuous Flow Networks with Average Cell Area of 1 million square feet

Table 55: Packing Factor Percentage for Continuous Flow Networks with Largest Cell Area of 1 million square feet

	Number of Lanes				
Network	1	2	3	4	5
Hexagon	95.7%	91.6%	87.8%	84.3%	80.9%
Square, Rectified Square	95.4%	91.0%	87.0%	83.2%	79.7%
Truncated Square	94.8%	89.9%	85.4%	81.2%	77.3%
Truncated Hexagon	95.0%	90.2%	85.8%	81.6%	77.6%
Truncated Trihexagon	95.3%	90.9%	86.7%	82.8%	79.2%
Martini	94.0%	88.4%	83.3%	78.6%	74.3%

Table 56: Packing Factor Percentage for Continuous Flow Networks with Average Cell Area of 1 million square feet

	Number of Lanes				
Network	1	2	3	4	5
Hexagon	95.7%	91.6%	87.8%	84.3%	80.9%
Square, Rectified Square	95.4%	91.0%	87.0%	83.2%	79.7%
Truncated Square	95.9%	92.0%	88.4%	85.0%	81.7%
Truncated Hexagon	96.3%	92.8%	89.5%	86.3%	83.2%
Truncated Trihexagon	96.1%	92.5%	89.0%	85.7%	82.6%
Martini	95.4%	91.1%	87.1%	83.3%	79.7%

6.2 Proposed Continuous Flow Intersection Results

6.2.1 Simulation Setup

The prior section highlighted several of the theoretical advantages and potential drawbacks of continuous flow networks. The benefits and drawbacks may be further

quantified through the use of microsimulation software. In order to simulate the proposed continuous flow networks and the appropriate baseline networks, the microscopic traffic simulator PARAMICS 6.9.3 was selected for assessing mobility metrics. In addition, EPA's MOVES emissions model was selected for assessing environmental metrics. Detailed information regarding the simulation software setup may be found in chapter 3. Prior to simulating the full continuous flow traffic networks, the networks were divided into individual continuous flow intersections. The divisions were based on facilitating comparison with isolated 4-leg intersections. Out of the 6 continuous flow traffic networks described in the preceding section, a single continuous flow traffic network was selected based on minimizing the expected increase in route distance relative to a conventional grid network. Next, a continuous flow intersection was extracted from the selected continuous flow traffic network and simulated in comparison to 3 baseline intersections: 1) a 3-lane conventional 4-leg intersection, 2) 3-lane round-about, and 3) a 2-lane Parallel Flow Intersection (PFI). In order to facilitate a fair comparison between the 4 intersections, the simulation area for each intersection was set as a fixed square of 2414 feet by 2414 feet. In addition, the speed limit for each intersection was set at 45 mph. Each simulation run was conducted for 1 hour, with additional time to permit all vehicles to exit the simulation.

6.2.2 PARAMICS Networks Description

6.2.2.1 Proposed UAID PARAMICS Networks

6.2.2.1.1 Rounded Truncated Square Intersection

Based on the values shown in the final column of Table 53, the truncated square tiling was selected as the most promising continuous flow network relative to a conventional signalized grid network. After accounting for the ratios of left-turn, right-turn, and through movements in a typical urban traffic scenario, the truncated square network provides the minimal increase in route distance relative to a conventional grid network for the continuous flow traffic networks considered. Isolating a single square of the truncated square network yields a UAID, as shown in Figure 142. Instead of using a perfect square surrounded by 4 regular octagons, the corners of the shapes are rounded in order to permit vehicles to travel at a higher speed than if the shapes were unrounded. The modified design with rounded polygons is shown in Figure 143. As depicted in part b of Figure 143, 2 lanes are reserved for the octagons, and 2 lanes are reserved for the rounded squares. As a result, the bi-directional links contain 2 lanes in each direction, and the unidirectional links contain 4 lanes in a single direction. Consequently, the average number of lanes in a given direction is 3.

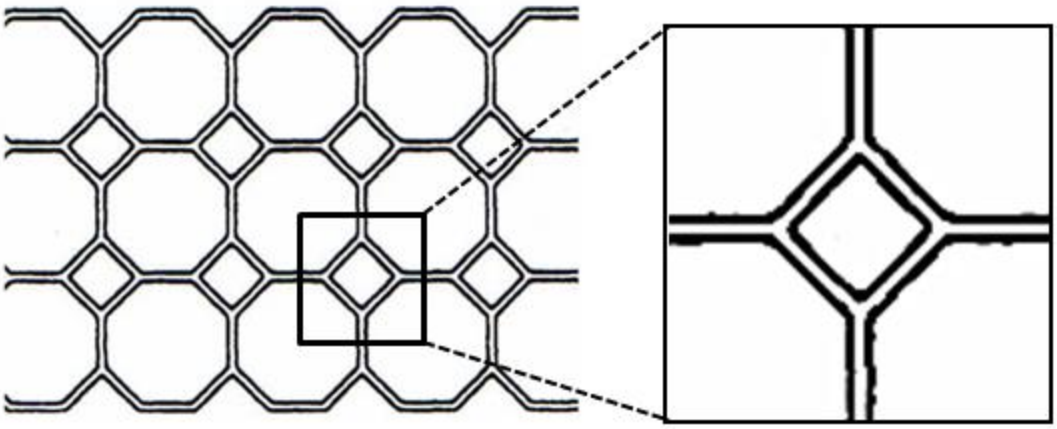

Figure 142: a) Truncated Square Traffic Network, b) Truncated Square UAID (in inset)

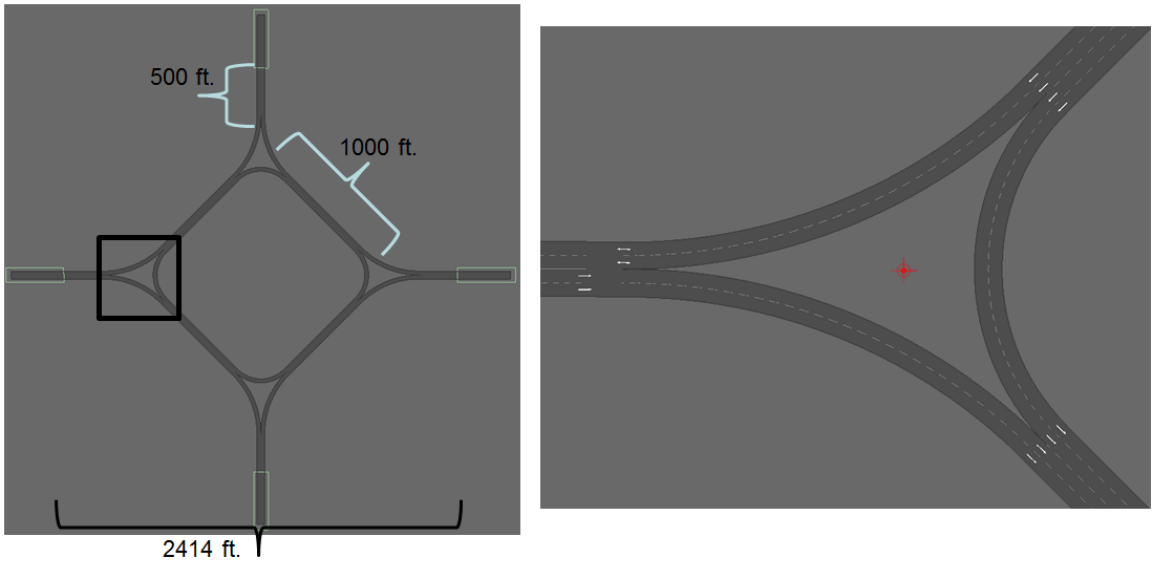

Figure 143: a) Rounded Truncated Square Design, b) West junction of rounded truncated square design (in inset)

The four sides of the rounded square are unidirectional links which circulate traffic in a counter clock-wise manner. In contrast, traffic flows clock-wise around the rounded octagons and the links separating adjacent octagons are bi-directional. The flow of traffic within a single rounded truncated square intersection is depicted in Figure 144

below. The individual traffic movements for traffic approaching the intersection are shown in Figure 145 below. The left-turn movement, through movement, and right-turn movement vehicles travel around 3, 2, and 1 sides of the square round-about, respectively. The left-turn and through movement vehicles make 2 lane changes in the intersection, once to enter the square round-about, and once to exit the square round-about. In contrast, the right-turn movement vehicles do not make any lane changes, and are effectively traveling one quarter of the distance around a rounded octagon. In terms of operations, there are no traffic signals, stop signs, or yield protocols placed in the PARAMICS network; instead, vehicles are expected to maintain speed within the 3-way junctions, (see Figure 143 part b), and throughout the entire intersection.

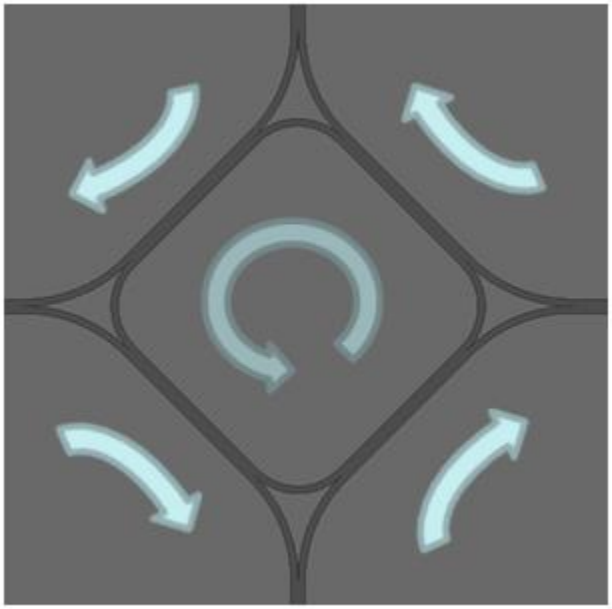

Figure 144: Direction of Traffic Flow within a rounded truncated square intersection

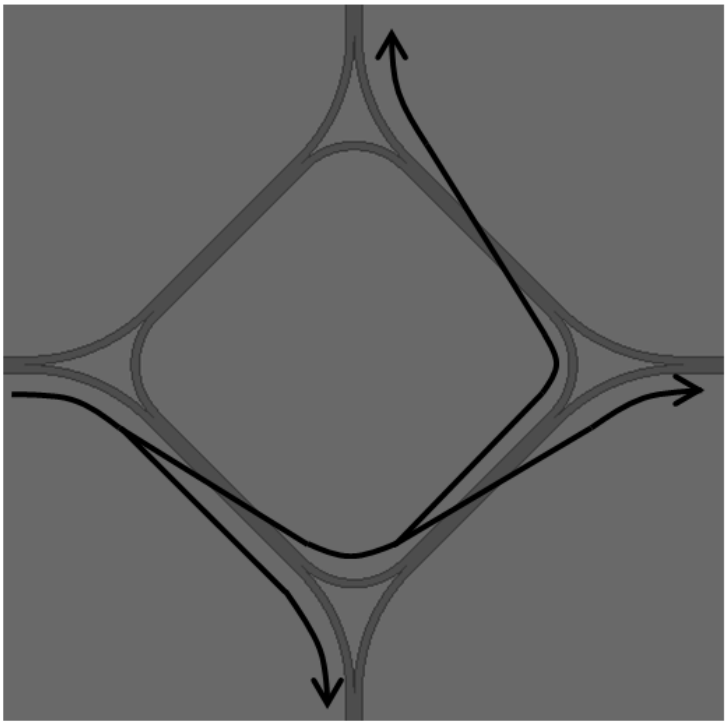

Figure 145: Rounded Truncated Square Intersection with West Approach Traffic Movements, Top to Bottom: 1) Left-turn movement, 2) Through movement, & 3) Right-turn movement

6.2.2.2 Baseline PARAMICS Networks

6.2.2.2.1 Conventional 4-leg Intersection

The first baseline selected was a 3-lane conventional 4-leg intersection. The PARAMICS network is shown in Figure 146 with the turning movements shown in Figure 147. The left-most lane is a dedicated left-turn lane. The middle lane is designated for through movement vehicles. Lastly, the right-most lane permits both through movement vehicles and right-turning vehicles. Right-turning vehicles are permitted to turn right on a red light. The 4 signal phases are the standard NEMA phases, as depicted in Figure 78. Fixed phase signal timing based on the Quick Estimation

Method (QEM) provided by FHWA was used to determine the cycle splits for a 120-second cycle.

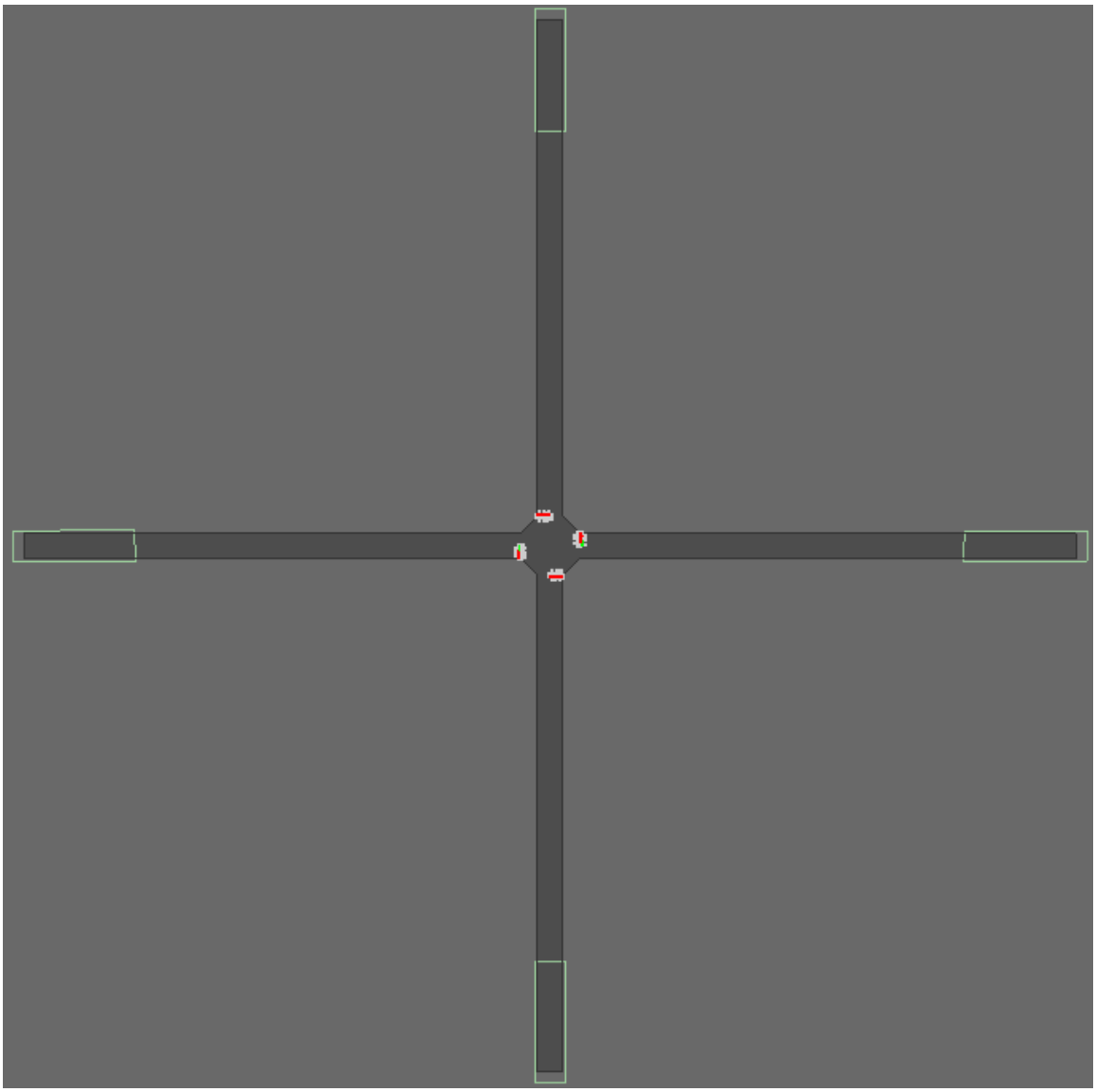

Figure 146: PARAMICS Network for 3-lane Conventional 4-leg Intersection

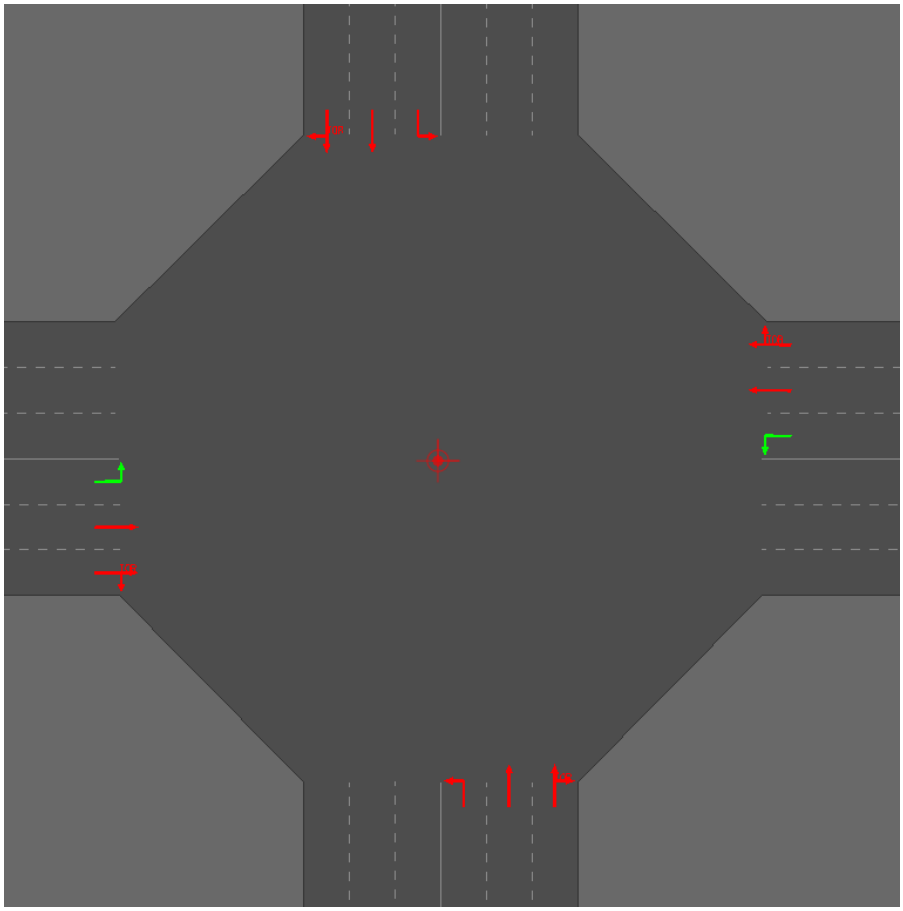

Figure 147: Turning Movements for Conventional 4-leg Intersection

6.2.2.2.2 Round-about Intersection

The second baseline selected was a 3-lane round-about intersection. The PARAMICS network is shown in Figure 148 with the turning movements shown in Figure 149. The round-about was designed with an inscribed diameter of 300 feet.

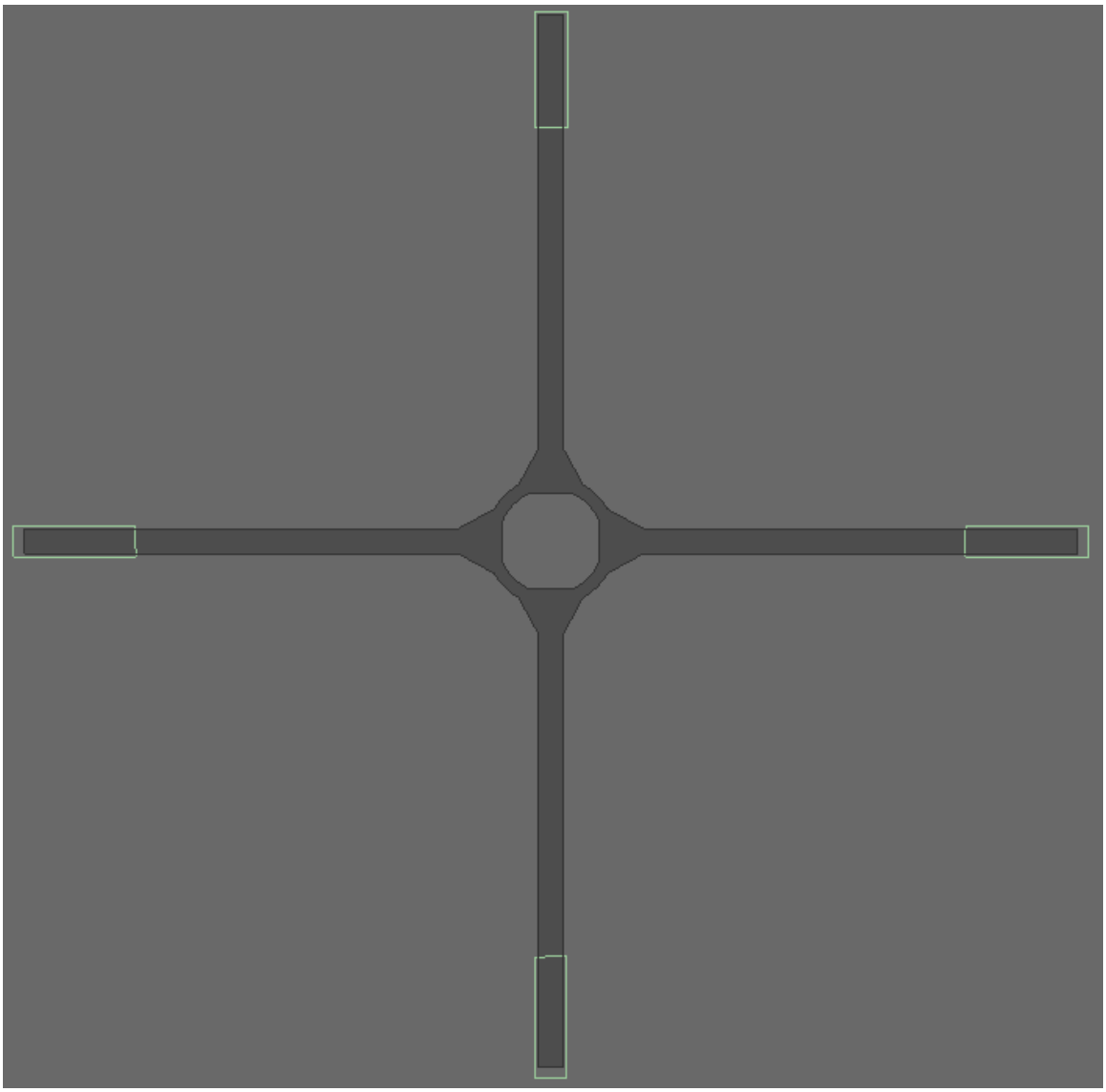

Figure 148: PARAMICS Network for 3-lane Round-about Intersection

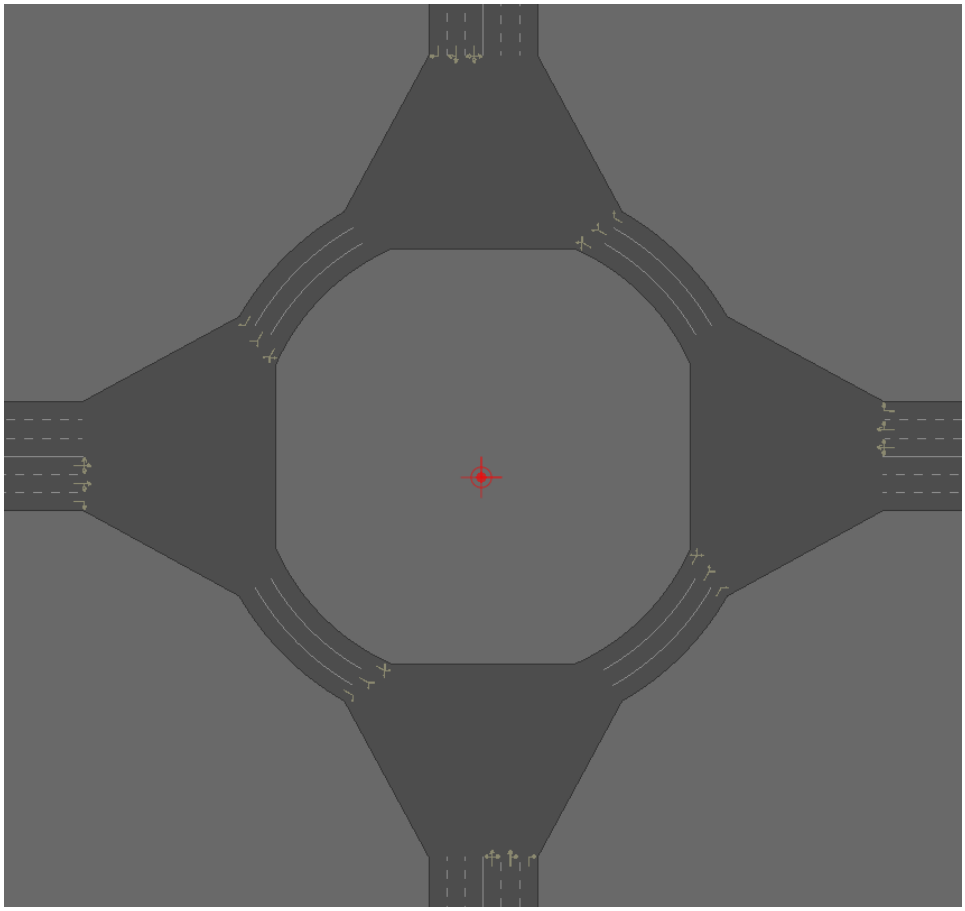

Figure 149: Turning Movements for Round-about Intersection

6.2.2.2.3 Parallel Flow Intersection (PFI)

The third baseline selected was a 2-lane Parallel Flow Intersection (PFI). The PARAMICS network is shown in Figure 150. There are only 2 lanes per leg permitting traffic to enter the overall intersection. Likewise, there are only 2 lanes per leg which permit traffic to exit the overall intersection. Accounting for the left-turn bays and the right-turn bays the intersection has a total of 5 lanes between the major and minor junctions. The average number of lanes for one direction of traffic is above 3 lanes per leg. Consequently, the rounded truncated square intersection, the conventional 4-leg

intersection, the round-about intersection, and the PFI each have 4 legs with an average number of lanes equal to or close to 3. The two signal phases are shown in Figures 151 and 152. The signal splits are determined by the distance between the major and minor junctions and the design speed limit. As a result, the PARAMICS network depicted in Figures 150-152 used a cycle length of 60 seconds with 30 seconds per signal phase. Each phase was comprised of 26 seconds of green time, 3 seconds of yellow time, and 1 second of red time.

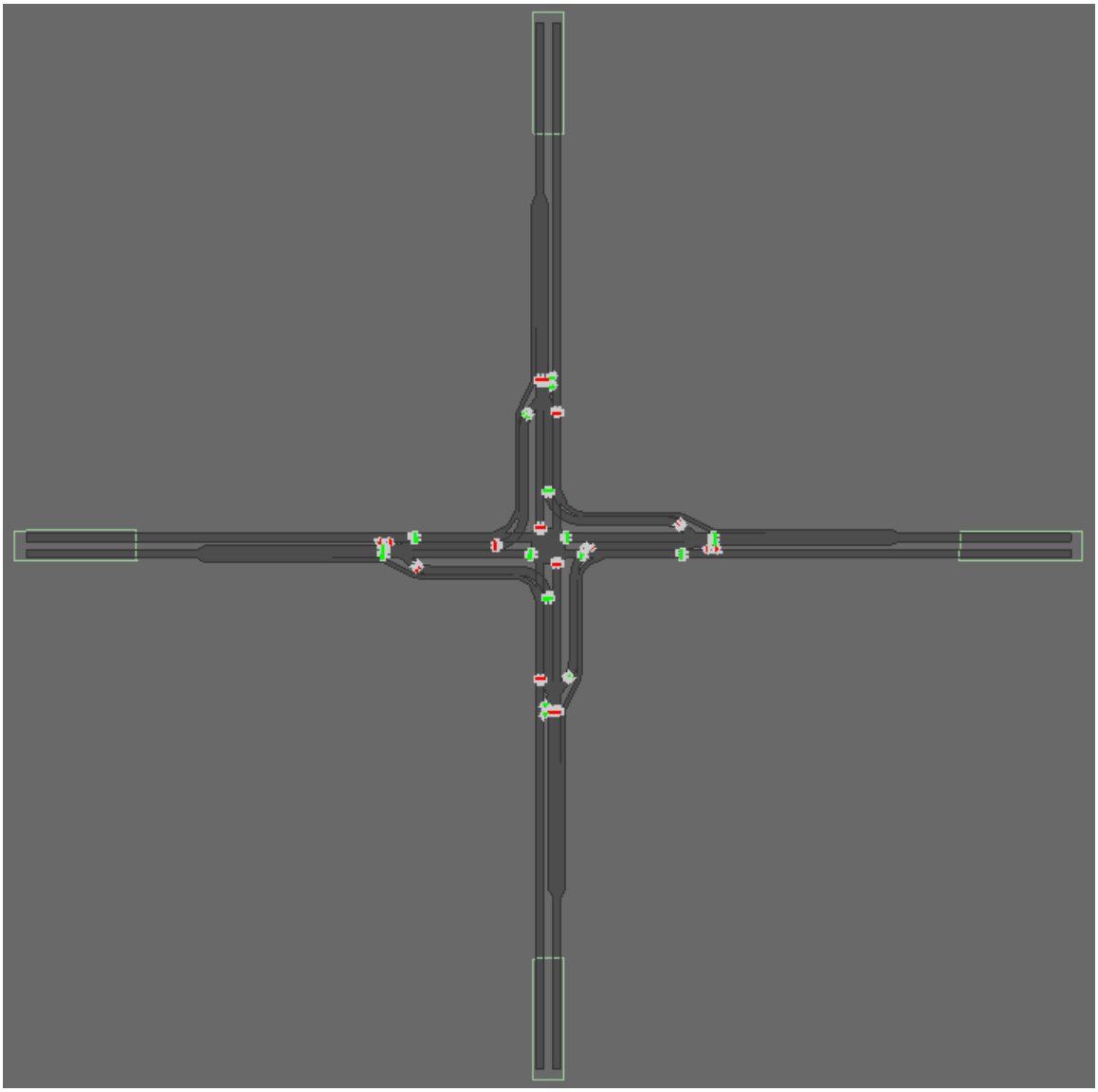

Figure 150: PARAMICS Network for 2-lane PFI

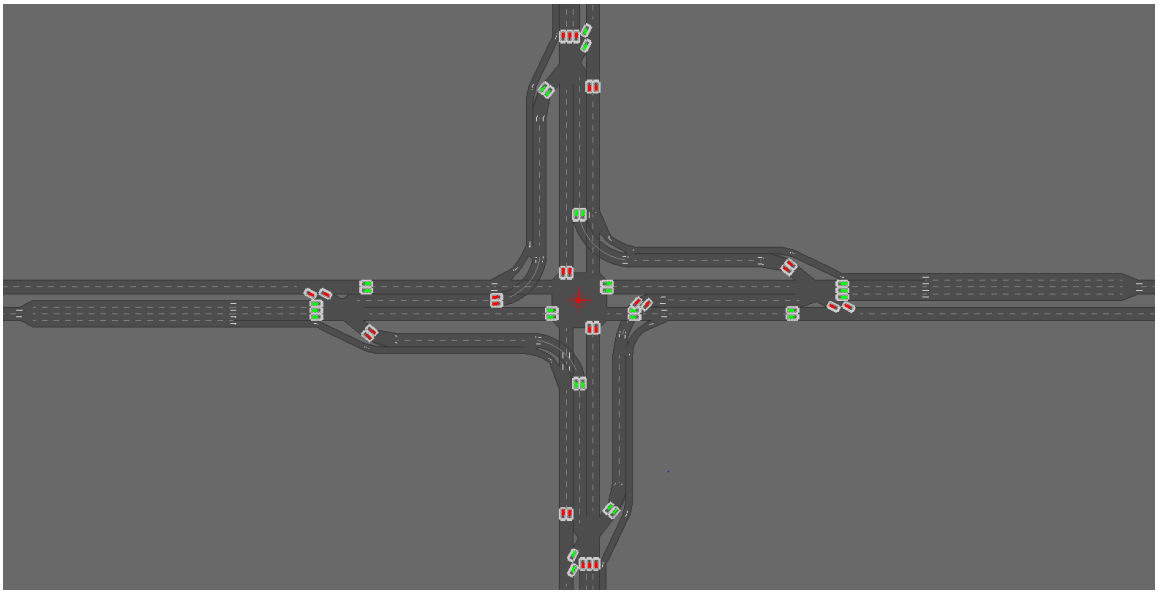

Figure 151: Phase 1 Signals for PFI

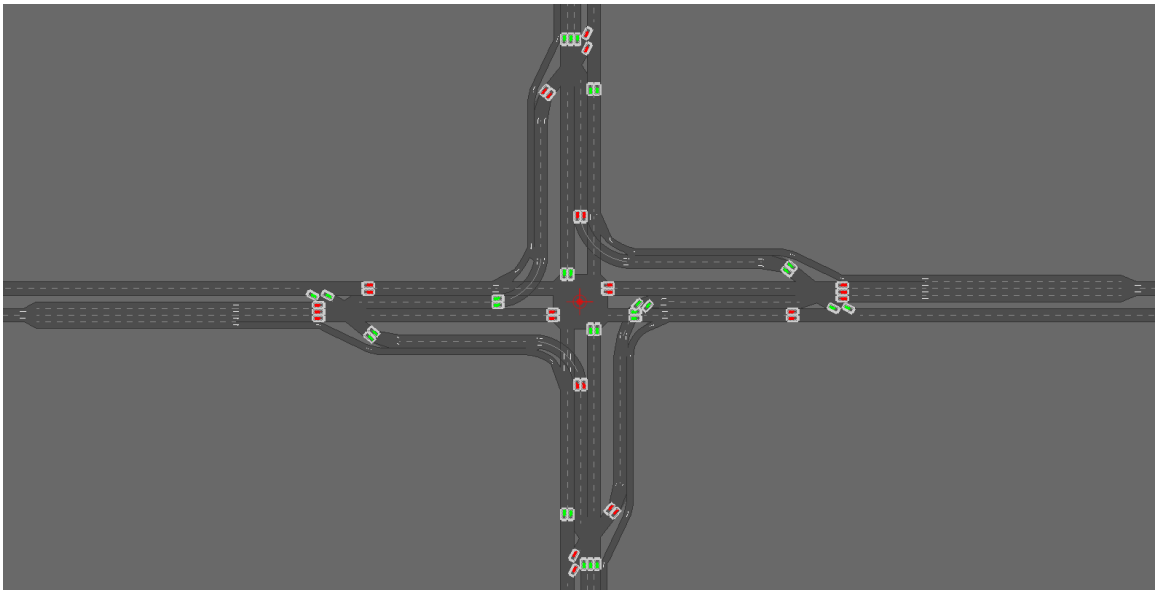

Figure 152: Phase 2 Signals for PFI

6.2.3 Volume Sensitivity Analysis

In the context of traffic intersection design, the purpose of sensitivity analyses is to determine an intersection design's sensitivity to a single parameter while keeping all other parameters constant. One of the most important sensitivity analyses for traffic systems is the traffic volume sensitivity analysis. The proposed continuous flow intersection, and the three baseline intersections were each simulated with the volume varied between 1000 and 6000 vehicles per hour (vph) in increments of 500 vph. The main street was set to have 50% more traffic volume than the cross-street. In addition, the ratios of left-turn movement, through movement, and right-turn movement traffic were fixed at 20%, 70%, and 10%, respectively. PARAMICS includes the option of specifying a demand profile. For a given volume of demand, the ratio of traffic arriving during a particular time interval of the simulation time duration may be varied. For example, a demand profile can specify that 20% of the overall hourly traffic arrive during the third 5-minute interval. For the following volume sensitivity analysis, the demand profile was kept constant. In order to produce a stronger baseline, the signal splits for the conventional 4-leg intersection were set based on *a priori* knowledge of the OD matrix, corresponding to the case of a perfectly tuned traffic light.

The following graphs include individual comparisons of the proposed rounded truncated square intersection versus each baseline intersection. For shorthand, the rounded truncated square intersection will be referred to as "Design 2." The volume sensitivity results are presented for mobility, energy, and emissions.

6.2.3.1 Proposed Design vs. Conventional 4-leg Intersection

Mobility results are shown in Figures 153-154. Energy results are shown in Figures 155-156. Lastly, emissions results are shown in Figure 157. Exact data values are presented in Tables 57-59. As shown in Figure 153, the average travel time for the truncated square intersection varies by less than 4 seconds from about 47 seconds at the lowest traffic volume to less than 51 seconds at 6000 vehicles per hour. In contrast, the average travel time for the conventional 4-leg intersection varies by 132 seconds from roughly 73 seconds at the lowest traffic volume to over 205 seconds at 6000 vehicles per hour. The savings in travel time of the truncated square intersection over the conventional 4-leg intersection varied between 35% at 1000 vehicles per hour to 75% at 6000 vehicles per hour. As stated earlier, a simple method of estimating an intersection's capacity is to find the volume at which the travel time is 80 seconds greater than the free flow travel time. A delay of greater than 80 seconds corresponds to a Level-Of-Service (LOS) of F, (see Table 1). Using the 80-second rule, the capacity of the conventional 4-leg intersection is around 5500 vehicles per hour. In contrast, the truncated square intersection maintains an LOS of A, (less than 10 seconds of delay), across traffic volumes from 1000 vehicles per hour to 6000 vehicles per hour. The capacity of the truncated square intersection is greater than 6000 vehicles per hour and will be shown in a subsequent section. It is worth noting that the percent improvement for average travel times for traffic volumes over 6000 vehicles per hour would be substantially greater than 75% due to the capacity of the conventional 4-leg intersection being exceeded.

In terms of energy, vehicles on average saved between 24% and 41% by using the truncated square intersection instead of the conventional 4-leg intersection. In addition, the emissions savings across the tested traffic volumes ranged between 19% and 63%. The primary reason for the energy savings is due to the nature of the continuous flow of traffic in the truncated square intersection versus the often interrupted flow of traffic in the conventional 4-leg intersection. Emissions savings are present because the vast majority of acceleration and deceleration events are either minimized in magnitude or eliminated. The energy used per vehicle for the truncated square intersection increases by about 16% from a traffic volume of 1000 vehicles per hour to 6000 vehicles per hour due to the increased energy expenditure for changing lanes with additional traffic. In contrast, the energy used per vehicle for the conventional 4-leg intersection increases by about 50% across the tested traffic volumes due to the capacity of the intersection being exceeded.

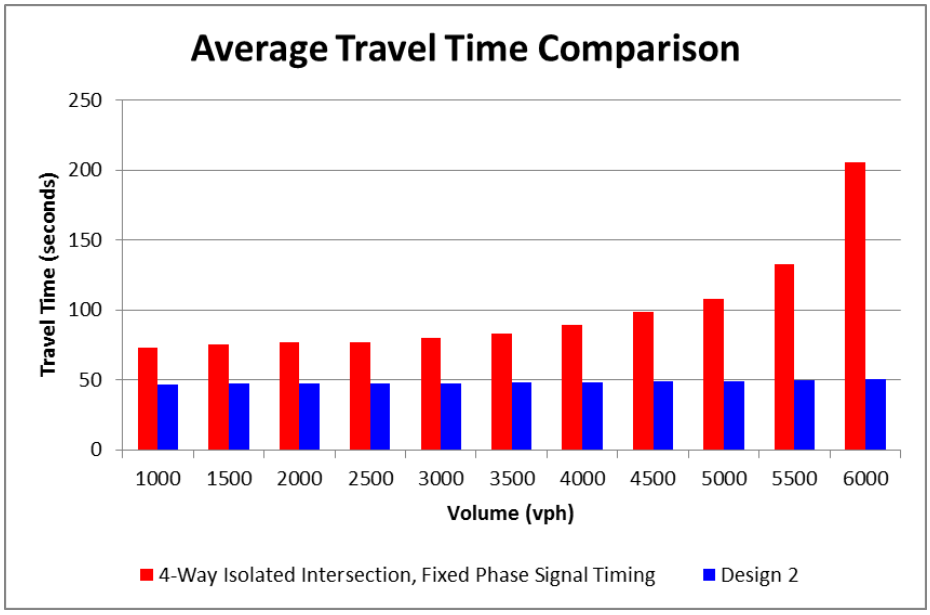

Figure 153: Average Travel Time Comparison of a Truncated Square Intersection vs. a Conventional 4-leg Intersection

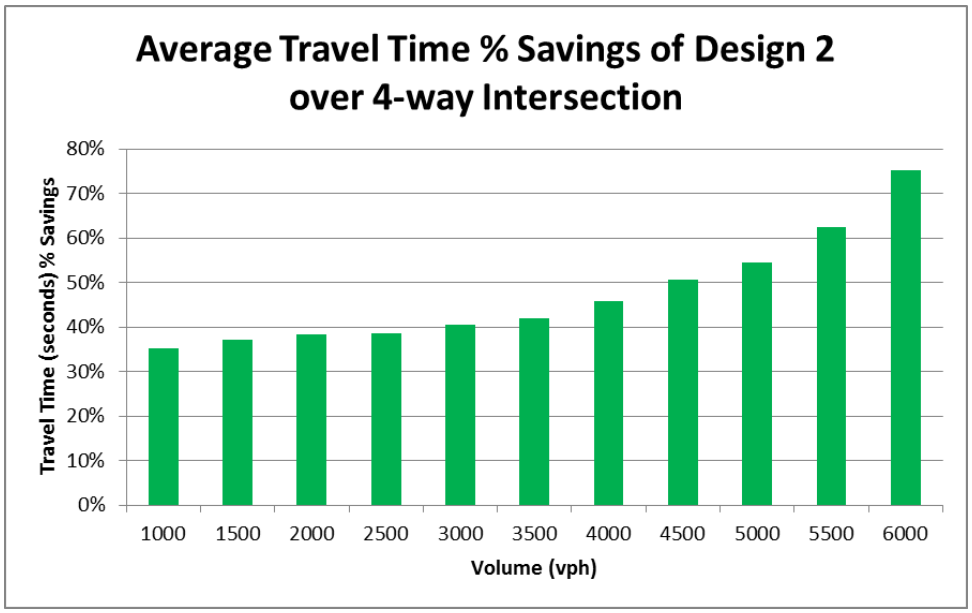

Figure 154: Average Travel Time Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection

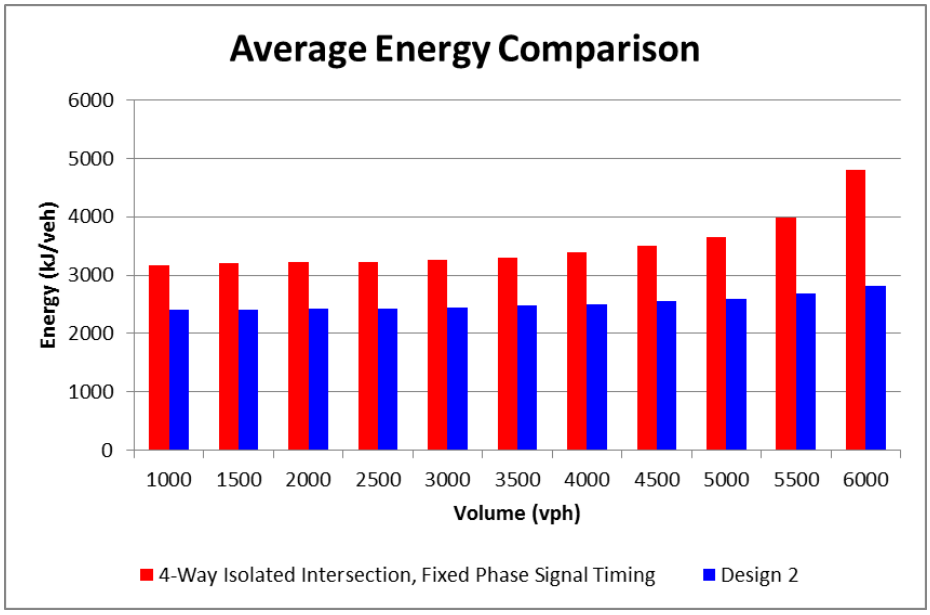

Figure 155: Average Energy Comparison of a Truncated Square Intersection vs. a Conventional 4-leg Intersection

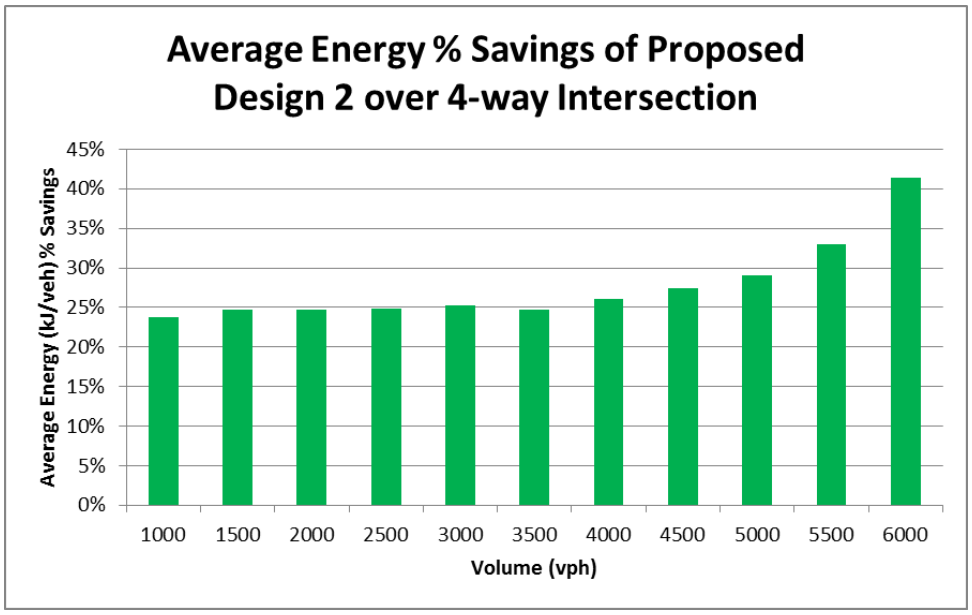

Figure 156: Average Energy Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection

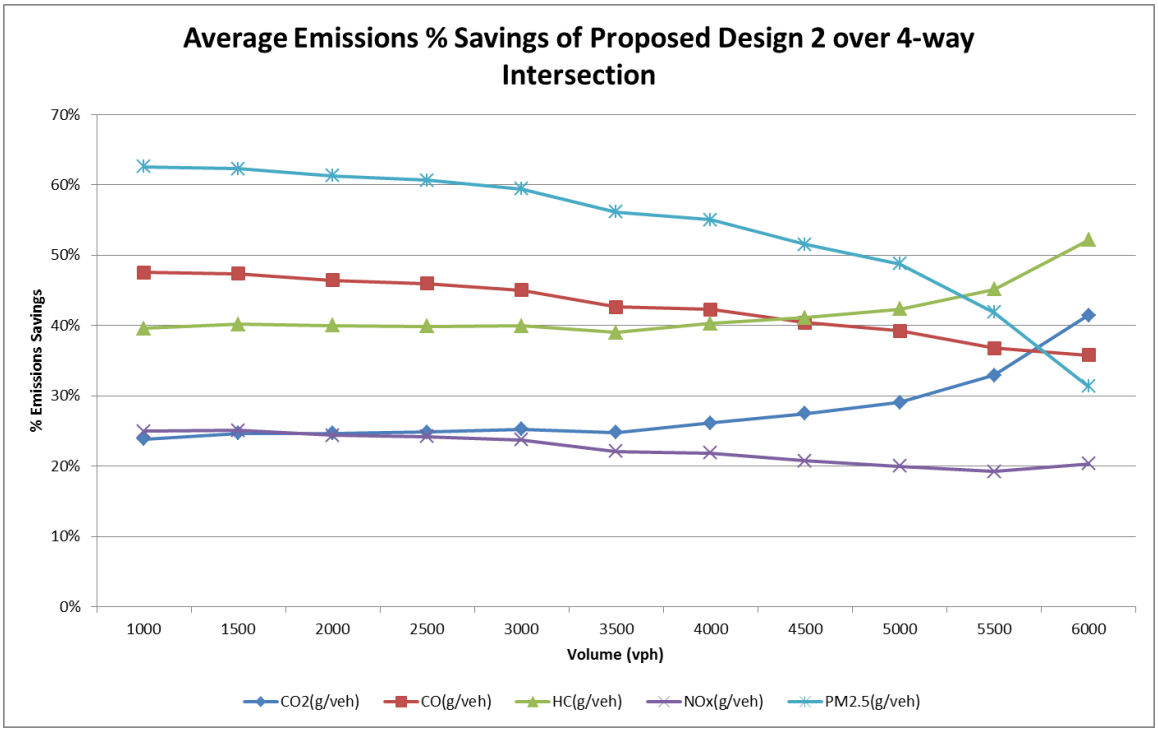

Figure 157: Average Emissions Savings of a Truncated Square Intersection over a Conventional 4-leg Intersection

Table 57: Conventional 4-leg Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	3170.4132	227.8566	4.8882	0.1555	0.5275	0.0355	72.9179
1500	3200.7972	230.0403	4.8896	0.1574	0.5276	0.0352	75.3797
2000	3217.6766	231.2534	4.8698	0.1585	0.5259	0.0349	77.3860
2500	3216.6993	231.1832	4.8490	0.1582	0.5251	0.0347	77.2489
3000	3261.9069	234.4322	4.8663	0.1608	0.5270	0.0345	80.4209
3500	3301.4029	237.2707	4.8773	0.1629	0.5293	0.0344	82.9094
4000	3392.5751	243.8232	4.9337	0.1685	0.5338	0.0343	89.5829
4500	3512.3050	252.4281	4.9616	0.1754	0.5378	0.0337	98.8051
5000	3645.9635	262.0341	5.0443	0.1833	0.5443	0.0337	108.1238
5500	3990.8813	286.8231	5.2042	0.2024	0.5646	0.0329	133.0343
6000	4800.4579	345.0068	5.6096	0.2481	0.6098	0.0316	205.4236

Table 58: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	2414.7912	173.5506	2.5629	0.0939	0.3957	0.0133	47.2282
1500	2410.3147	173.2288	2.5720	0.0941	0.3952	0.0133	47.3024
2000	2422.8701	174.1312	2.6090	0.0951	0.3978	0.0135	47.5950
2500	2417.6074	173.7529	2.6203	0.0951	0.3979	0.0136	47.4190
3000	2438.5764	175.2600	2.6759	0.0966	0.4019	0.0140	47.8483
3500	2483.3274	178.4762	2.7964	0.0993	0.4123	0.0151	48.0957
4000	2505.7684	180.0890	2.8471	0.1006	0.4172	0.0154	48.4110
4500	2546.4025	183.0094	2.9577	0.1032	0.4263	0.0163	48.7907
5000	2586.0940	185.8620	3.0646	0.1057	0.4353	0.0172	49.0973
5500	2676.4253	192.3541	3.2895	0.1110	0.4559	0.0191	49.7147
6000	2811.0443	202.0291	3.6004	0.1187	0.4857	0.0217	50.8981

Table 59: % Improvement of Truncated Square Intersection over Conventional 4-leg Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	23.83%	23.83%	47.57%	39.61%	24.98%	62.57%	35.23%
1500	24.70%	24.70%	47.40%	40.20%	25.10%	62.31%	37.25%
2000	24.70%	24.70%	46.43%	40.02%	24.36%	61.28%	38.50%
2500	24.84%	24.84%	45.96%	39.87%	24.21%	60.67%	38.62%
3000	25.24%	25.24%	45.01%	39.93%	23.74%	59.41%	40.50%
3500	24.78%	24.78%	42.67%	39.03%	22.11%	56.17%	41.99%
4000	26.14%	26.14%	42.29%	40.29%	21.86%	55.04%	45.96%
4500	27.50%	27.50%	40.39%	41.14%	20.74%	51.55%	50.62%
5000	29.07%	29.07%	39.25%	42.34%	20.02%	48.76%	54.59%
5500	32.94%	32.94%	36.79%	45.15%	19.25%	41.89%	62.63%
6000	41.44%	41.44%	35.82%	52.16%	20.35%	31.33%	75.22%

6.2.3.2 Proposed Design vs. Round-about Intersection

Mobility results are shown in Figures 158-159. Energy results are shown in Figures 160-161. Lastly, emissions results are shown in Figure 162. Exact data values

are presented in Tables 60-62. As shown in Figure 158, the average travel time for the round-about varied between 60 and 945 seconds at 1000 and 6000 vehicles per hour, respectively. Based on the 80-second rule, the capacity of the round-about is a little over 3500 vehicles per hour. The travel times for the round-about increase significantly when the capacity of the round-about is exceeded. As a result, the travel time savings of the truncated square intersection over the round-about varied from 21% at 1000 vehicles per hour to 95% at 6000 vehicles per hour. Even under free flow conditions, the proposed continuous flow intersection still provides a 20% savings in travel time. Although vehicles are traveling a shorter distance in a round-about relative to the proposed intersection, vehicles must still slow down due to the yield protocol of the round-about. In contrast, the only source of vehicles slowing down in the continuous flow truncated square intersection is due to vehicles exchanging lanes in heavy traffic.

In terms of energy, vehicles on average saved between 28% and 54% by using the truncated square intersection instead of the round-about. In addition, the emissions savings across the tested traffic volumes ranged between 28% and 66%. The increase in average energy expenditure per vehicle across the tested volumes was roughly 78% for the round-about, and only 16% for the truncated square intersection. A majority of the energy increasing with traffic volume for the round-about is due to the capacity of the round-about being exceeded.

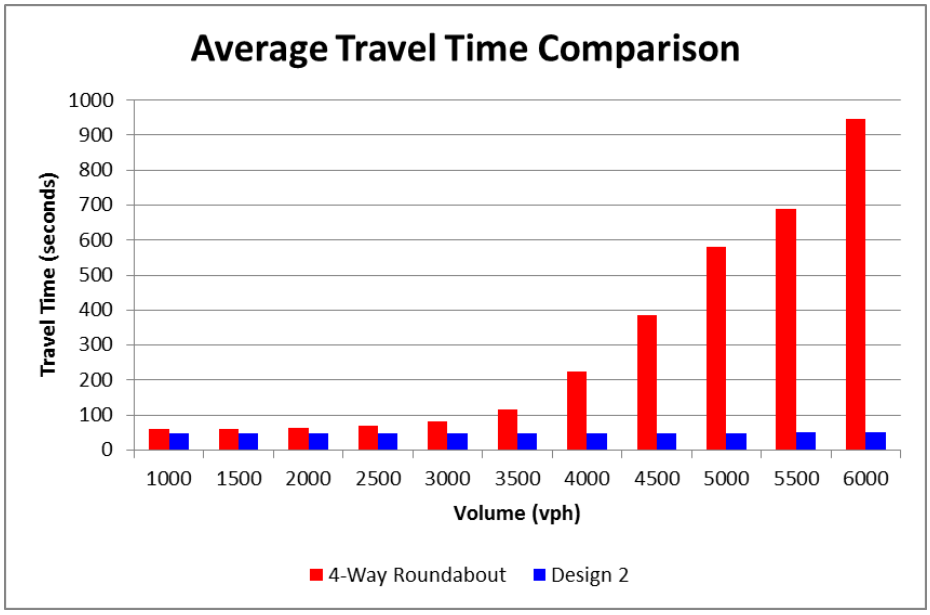

Figure 158: Average Travel Time Comparison of a Truncated Square Intersection vs. a Round-about Intersection

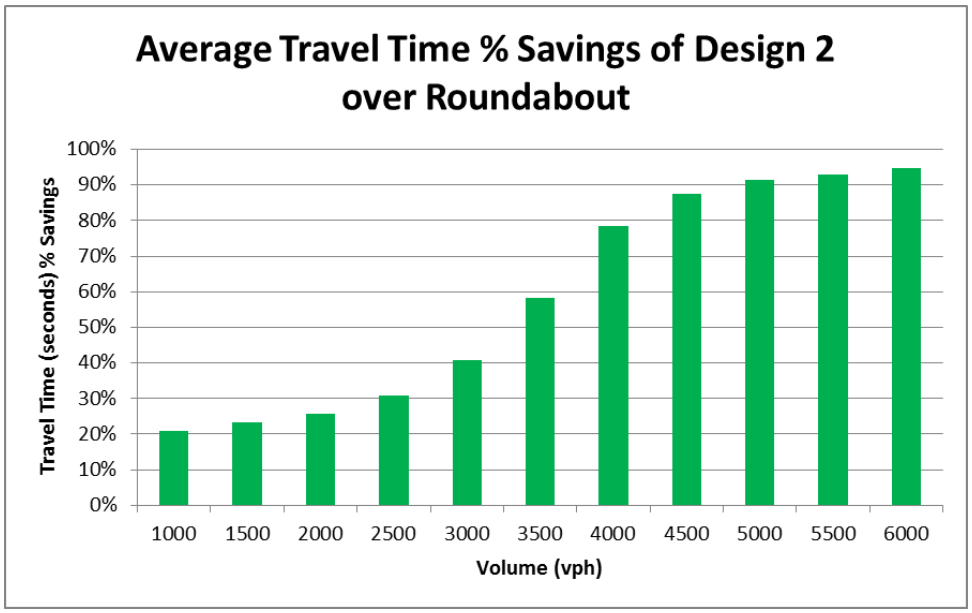

Figure 159: Average Travel Time Savings of a Truncated Square Intersection over a Round-about Intersection

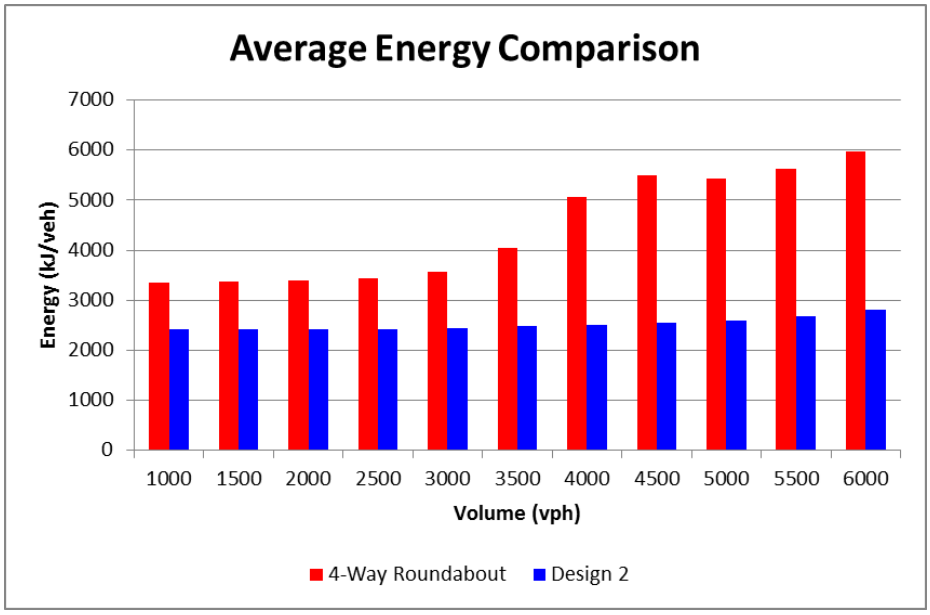

Figure 160: Average Energy Comparison of a Truncated Square Intersection vs. a Round-about Intersection

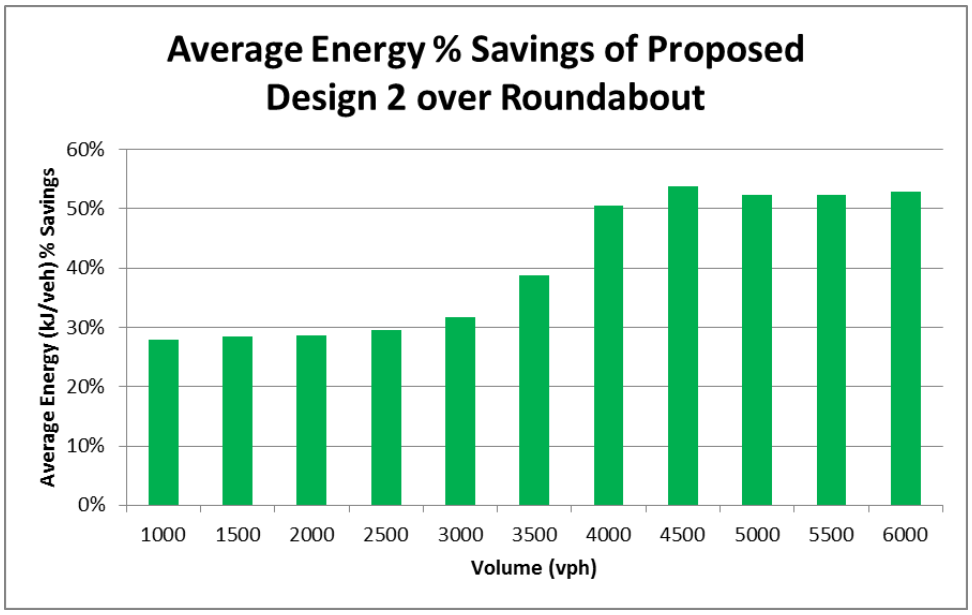

Figure 161: Average Energy Savings of a Truncated Square Intersection over a Round-about Intersection

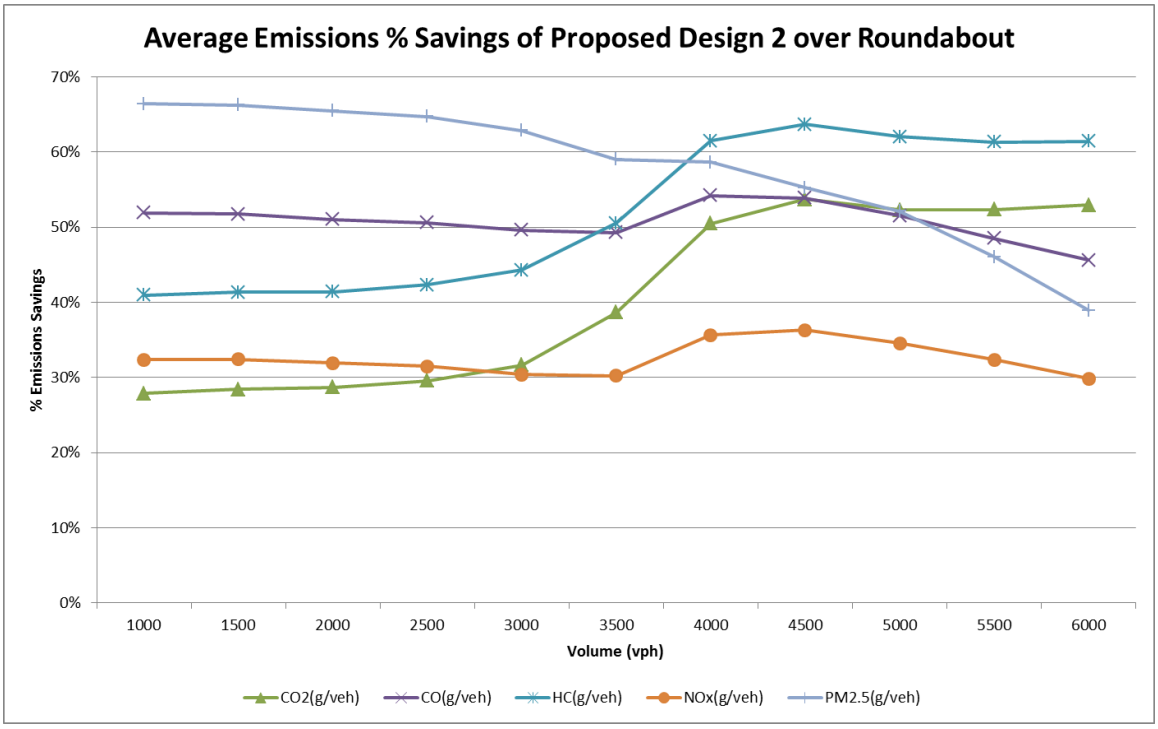

Figure 162: Average Emissions Savings of a Truncated Square Intersection over a Round-about Intersection

Table 60: Round-about Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	3349.3489	240.7169	5.3314	0.1591	0.5851	0.0395	59.6714
1500	3368.5615	242.0977	5.3310	0.1605	0.5847	0.0394	61.6875
2000	3397.4123	244.1712	5.3288	0.1624	0.5842	0.0392	64.1696
2500	3432.9524	246.7254	5.3058	0.1649	0.5809	0.0387	68.4917
3000	3569.1548	256.5142	5.3099	0.1734	0.5777	0.0378	80.6311
3500	4050.4848	291.1071	5.5172	0.2009	0.5911	0.0368	115.2393
4000	5063.0558	363.8798	6.2178	0.2615	0.6483	0.0373	225.7775
4500	5501.8957	395.4189	6.4153	0.2844	0.6694	0.0365	385.7126
5000	5421.5053	389.6414	6.3246	0.2785	0.6651	0.0360	579.7619
5500	5619.1373	403.8451	6.3916	0.2873	0.6742	0.0355	688.7798
6000	5972.9620	429.2742	6.6223	0.3078	0.6921	0.0356	944.9357

Table 61: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	2414.7912	173.5506	2.5629	0.0939	0.3957	0.0133	47.2282
1500	2410.3147	173.2288	2.5720	0.0941	0.3952	0.0133	47.3024
2000	2422.8701	174.1312	2.6090	0.0951	0.3978	0.0135	47.5950
2500	2417.6074	173.7529	2.6203	0.0951	0.3979	0.0136	47.4190
3000	2438.5764	175.2600	2.6759	0.0966	0.4019	0.0140	47.8483
3500	2483.3274	178.4762	2.7964	0.0993	0.4123	0.0151	48.0957
4000	2505.7684	180.0890	2.8471	0.1006	0.4172	0.0154	48.4110
4500	2546.4025	183.0094	2.9577	0.1032	0.4263	0.0163	48.7907
5000	2586.0940	185.8620	3.0646	0.1057	0.4353	0.0172	49.0973
5500	2676.4253	192.3541	3.2895	0.1110	0.4559	0.0191	49.7147
6000	2811.0443	202.0291	3.6004	0.1187	0.4857	0.0217	50.8981

Table 62: % Improvement of Truncated Square Intersection over Round-about Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	27.90%	27.90%	51.93%	40.95%	32.37%	66.42%	20.85%
1500	28.45%	28.45%	51.75%	41.37%	32.40%	66.27%	23.32%
2000	28.68%	28.68%	51.04%	41.44%	31.91%	65.49%	25.83%
2500	29.58%	29.58%	50.61%	42.32%	31.50%	64.73%	30.77%
3000	31.68%	31.68%	49.61%	44.29%	30.43%	62.89%	40.66%
3500	38.69%	38.69%	49.32%	50.57%	30.26%	59.04%	58.26%
4000	50.51%	50.51%	54.21%	61.52%	35.65%	58.68%	78.56%
4500	53.72%	53.72%	53.90%	63.69%	36.32%	55.28%	87.35%
5000	52.30%	52.30%	51.55%	62.05%	34.55%	52.15%	91.53%
5500	52.37%	52.37%	48.53%	61.37%	32.38%	46.08%	92.78%
6000	52.94%	52.94%	45.63%	61.45%	29.83%	38.96%	94.61%

6.2.3.3 Proposed Design vs. Parallel Flow Intersection

Mobility results are shown in Figures 163-164. Energy results are shown in Figures 165-166. Lastly, emissions results are shown in Figure 167. Exact data values

are presented in Tables 63-65. The previous two baseline comparisons tested volume up to 6000 vehicles per hour, at which point, the capacities of the conventional 4-leg intersection and round-about were exceeded. For the baseline comparison of the truncated square intersection versus the PFI, the highest test volume was gradually increased until the capacity of either the truncated square intersection, or the PFI was reached. As a result, the results below are included up to a maximum volume of 7500 vehicles per hour. Using the 80-second rule, the capacity of the truncated square intersection is around 7500 vehicles per hour. The capacity of the PFI is greater than 7500 vehicles per hour. Although the PFI has a higher capacity than a truncated square intersection, the latter provides average travel time savings from between 27% and 33% for traffic volumes less than 7500 vehicles per hour. At a traffic volume of 7500 vehicles per hour, the PFI intersection provides a travel time savings of 25% over the truncated square intersection. The travel time benefit for truncated square intersection is negative at 7500 vehicles per hour due to traffic operating at saturated conditions for the truncated square intersection and under-saturated conditions for the PFI.

In terms of energy, vehicles on average saved between 10% and 33% by using the truncated square intersection instead of the PFI for traffic volumes less than 7500 vehicle per hour. When the truncated square intersection is at capacity, 7500 vehicle per hour, the energy penalty is -8%. In addition, the emissions savings up to 7000 vehicles per hour ranged between 7% and 74%. At 7500 vehicles per hour, the emissions savings for CO_2 and NO_X were slightly negative. The emissions savings of the truncated square

intersection over the PFI decrease as the traffic volume is increased with the savings essentially disappearing when the capacity of the truncated square intersection is reached.

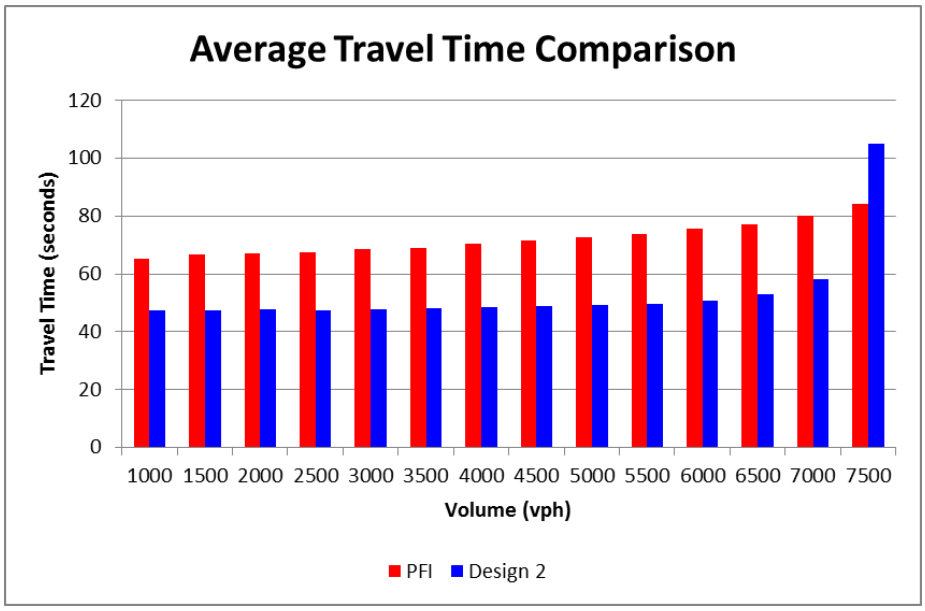

Figure 163: Average Travel Time Comparison of a Truncated Square Intersection vs. a PFI

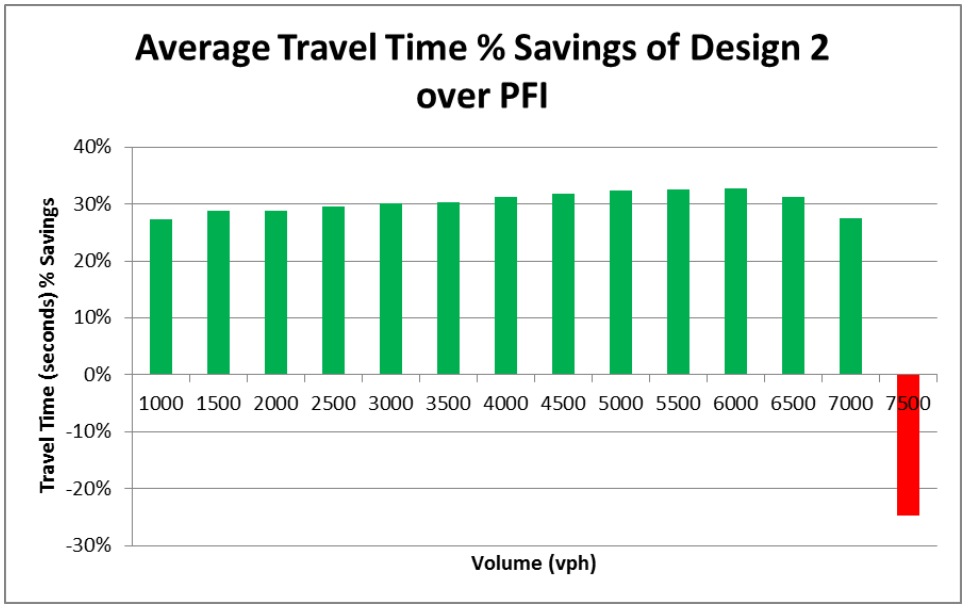

Figure 164: Average Travel Time Savings of a Truncated Square Intersection over a PFI

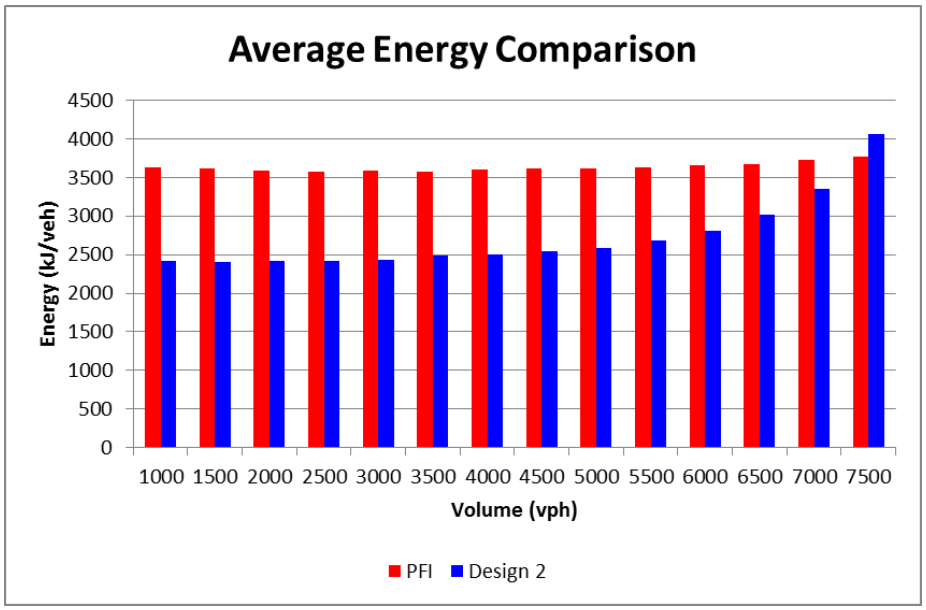

Figure 165: Average Energy Comparison of a Truncated Square Intersection vs. a PFI

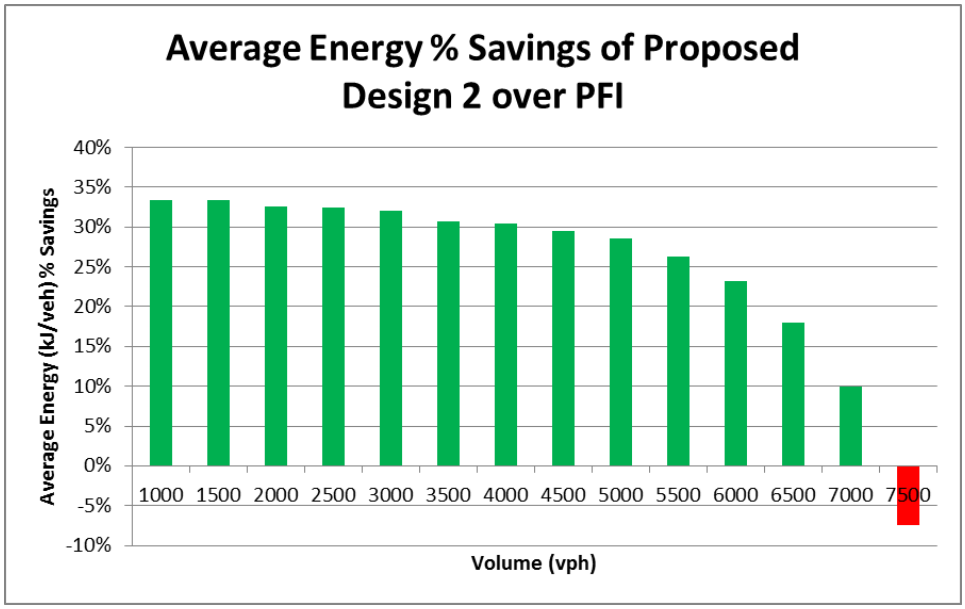

Figure 166: Average Energy Savings of a Truncated Square Intersection over a PFI

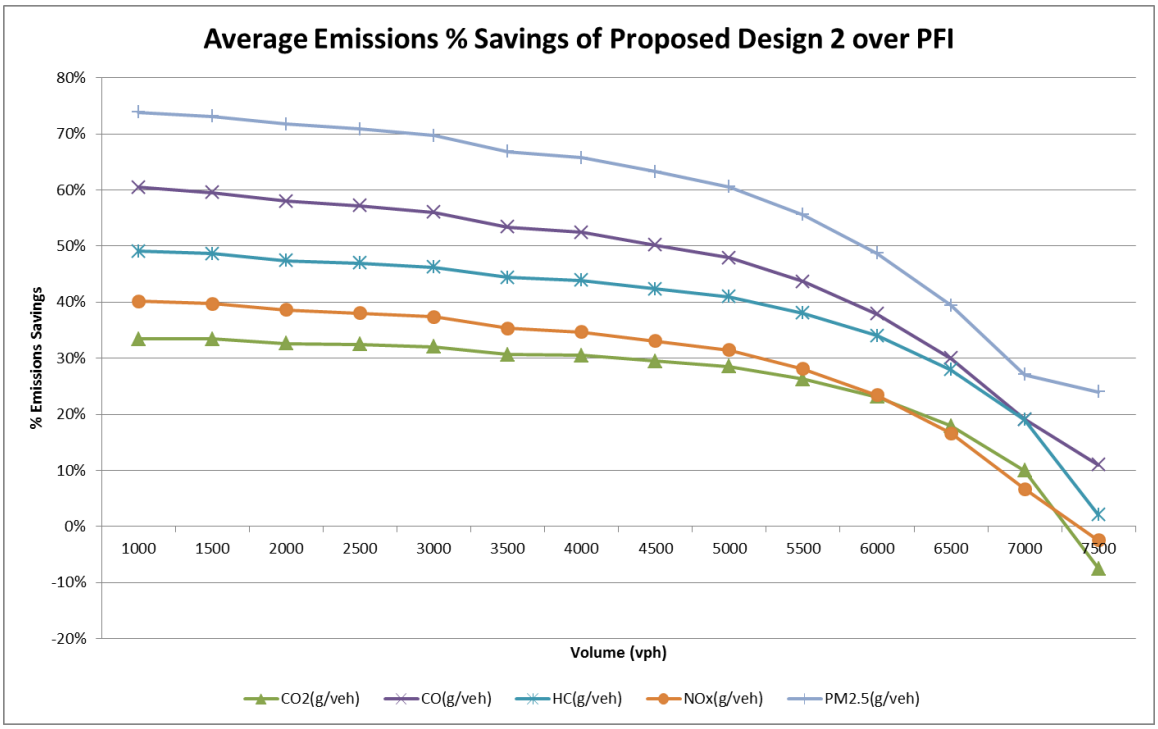

Figure 167: Average Emissions Savings of a Truncated Square Intersection over a PFI

Table 63: PFI, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	*Energy (kJ/veh)*	*CO2 (g/veh)*	*CO (g/veh)*	*HC (g/veh)*	*NOx (g/veh)*	*PM2.5 (g/veh)*	*VHT (s/veh)*
1000	3626.1114	260.6075	6.4869	0.1844	0.6615	0.0508	65.0578
1500	3620.3120	260.1907	6.3585	0.1832	0.6559	0.0494	66.5444
2000	3595.0287	258.3736	6.2132	0.1808	0.6481	0.0478	66.9206
2500	3579.1283	257.2308	6.1184	0.1794	0.6423	0.0468	67.4186
3000	3589.7370	257.9933	6.0832	0.1796	0.6416	0.0463	68.5311
3500	3580.7196	257.3452	6.0002	0.1785	0.6374	0.0454	69.1044
4000	3604.8304	259.0780	5.9880	0.1793	0.6386	0.0450	70.4708
4500	3611.0977	259.5284	5.9340	0.1791	0.6367	0.0444	71.5209
5000	3618.3351	260.0486	5.8850	0.1790	0.6349	0.0437	72.6563
5500	3631.5508	260.9984	5.8402	0.1791	0.6342	0.0431	73.7519
6000	3657.8157	262.8861	5.7975	0.1798	0.6340	0.0424	75.8139
6500	3671.6304	263.8789	5.7796	0.1803	0.6336	0.0421	76.9438
7000	3722.0484	267.5024	5.7520	0.1821	0.6357	0.0414	80.1504
7500	3775.5945	271.3508	5.7178	0.1845	0.6359	0.0406	84.2828

Table 64: Truncated Square Intersection, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	2414.7912	173.5506	2.5629	0.0939	0.3957	0.0133	47.2282
1500	2410.3147	173.2288	2.5720	0.0941	0.3952	0.0133	47.3024
2000	2422.8701	174.1312	2.6090	0.0951	0.3978	0.0135	47.5950
2500	2417.6074	173.7529	2.6203	0.0951	0.3979	0.0136	47.4190
3000	2438.5764	175.2600	2.6759	0.0966	0.4019	0.0140	47.8483
3500	2483.3274	178.4762	2.7964	0.0993	0.4123	0.0151	48.0957
4000	2505.7684	180.0890	2.8471	0.1006	0.4172	0.0154	48.4110
4500	2546.4025	183.0094	2.9577	0.1032	0.4263	0.0163	48.7907
5000	2586.0940	185.8620	3.0646	0.1057	0.4353	0.0172	49.0973
5500	2676.4253	192.3541	3.2895	0.1110	0.4559	0.0191	49.7147
6000	2811.0443	202.0291	3.6004	0.1187	0.4857	0.0217	50.8981
6500	3012.0550	216.4756	4.0475	0.1299	0.5283	0.0255	52.8838
7000	3350.2713	240.7831	4.6532	0.1475	0.5930	0.0302	58.0220
7500	4058.9460	291.7154	5.0884	0.1806	0.6516	0.0308	105.0490

Table 65: % Improvement of Truncated Square Intersection over PFI, Traffic Volume Sensitivity Analysis Results

Volume (vphpi)	Energy (kJ/veh)	CO2 (g/veh)	CO (g/veh)	HC (g/veh)	NOx (g/veh)	PM2.5 (g/veh)	VHT (s/veh)
1000	33.41%	33.41%	60.49%	49.07%	40.18%	73.87%	27.41%
1500	33.42%	33.42%	59.55%	48.64%	39.75%	73.08%	28.92%
2000	32.60%	32.60%	58.01%	47.42%	38.62%	71.74%	28.88%
2500	32.45%	32.45%	57.17%	46.97%	38.04%	70.87%	29.66%
3000	32.07%	32.07%	56.01%	46.23%	37.36%	69.71%	30.18%
3500	30.65%	30.65%	53.39%	44.36%	35.32%	66.80%	30.40%
4000	30.49%	30.49%	52.45%	43.89%	34.68%	65.76%	31.30%
4500	29.48%	29.48%	50.16%	42.36%	33.05%	63.21%	31.78%
5000	28.53%	28.53%	47.93%	40.96%	31.43%	60.56%	32.43%
5500	26.30%	26.30%	43.67%	38.01%	28.12%	55.56%	32.59%
6000	23.15%	23.15%	37.90%	33.99%	23.39%	48.72%	32.86%
6500	17.96%	17.96%	29.97%	27.95%	16.63%	39.41%	31.27%
7000	9.99%	9.99%	19.10%	19.00%	6.71%	27.10%	27.61%
7500	-7.50%	-7.50%	11.01%	2.10%	-2.46%	24.04%	-24.64%

6.2.3.4 Proposed Design vs. all 3 Baselines

In addition to the individually comparing the truncated square intersection with each baseline intersection, a similar comparison can be made by simultaneously plotting the results of the truncated square intersection and the 3 baseline intersections. The mobility results comparing all 4 intersections are shown in Figure 168. The energy results are shown in Figure 169. Based on travel time results and applying the 80-second rule, the capacities for the round-about, conventional 4-leg intersection, truncated square intersection, and PFI were estimated as 3500 vehicles per hour, 5500 vehicles per hour, 7500 vehicles per hour, and 8500 vehicles per hour, respectively. As can be observed in Figure 168, the travel time arcs up as the traffic volume is increased. At relatively low traffic volumes, small increases in volume lead to a nearly horizontal trend for travel times. In contrast, if the volume is set to match the capacity of a given intersection, even small increases in volume lead to a nearly vertical trend for travel times.

A similar set of trends may be observed for the average energy comparison of the 4 intersections. At relatively low traffic volumes, small increases in volume lead to a nearly horizontal trend for travel times. As the traffic volume is increased, the average energy arcs up until the intersection reaches its capacity. When the capacity of the intersection is exceeded, energy consumption continues to increase; however, the increase is not exponential. Therefore, although the average travel time of a given intersection may be expressed as an exponential function of volume, the average energy

of a given intersection can only be expressed as an exponential function of volume for volumes that are not over-saturated.

In addition to travel time, the average delay time for each intersection is shown in Figure 170. The delay times for each intersection can be classified into Levels Of Service (LOS). The conventional 4-leg intersection has an LOS of D at 1000 vph, and reaches an LOS of F by 5500 vph. The round-about has an LOS of B at 1000 vph, but reaches an LOS of F by 4000 vph. The PFI has an LOS of C at 1000 vph and does not reach an LOS of F until a volume of 8500 vph. The truncated square intersection has an LOS of A from 1000 vph to 6500 vph, an LOS of B at 7000 vph, and an LOS of E at 7500 vph. Out of the four intersections tested, the truncated square intersection was the only intersection to achieve an LOS of A. Incredibly, the truncated square intersection achieved an LOS of A across a large range of volumes, including at volumes which exceeded the capacities of the round-about and the conventional 4-leg intersection. At lower traffic volumes, there is virtually no delay for vehicles traveling through the truncated square intersection.

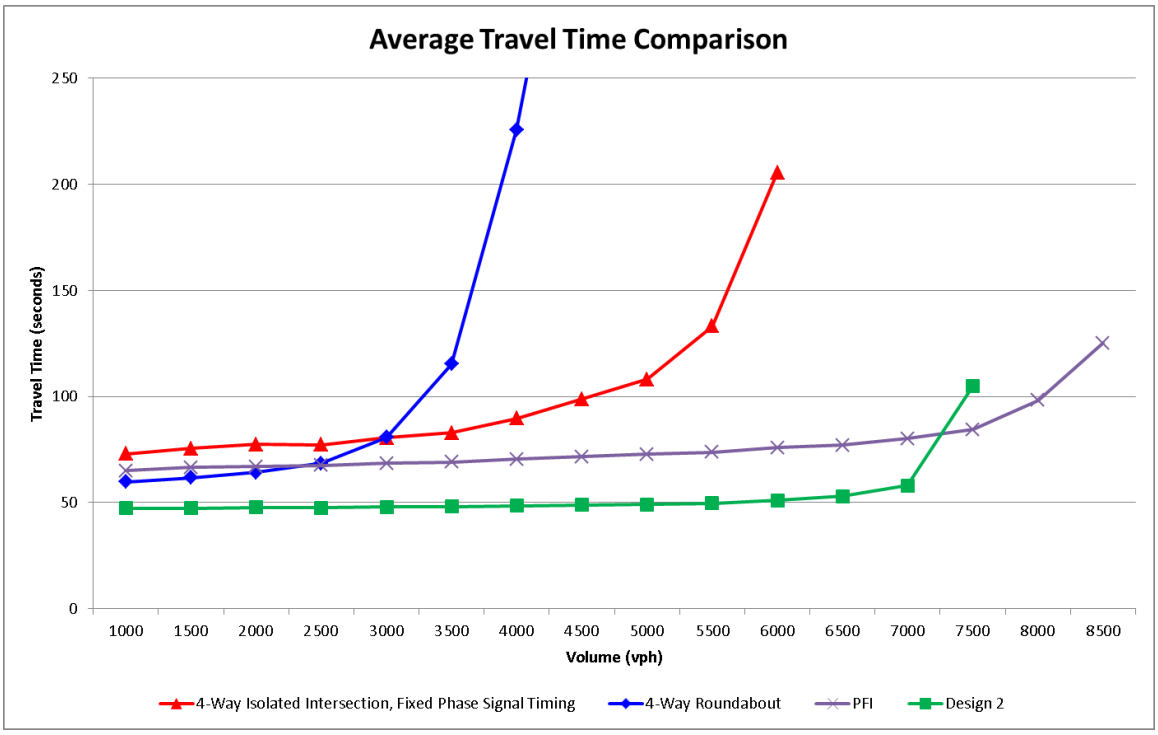

Figure 168: Average Travel Time Comparison of Truncated Square Intersection and Baseline Intersections

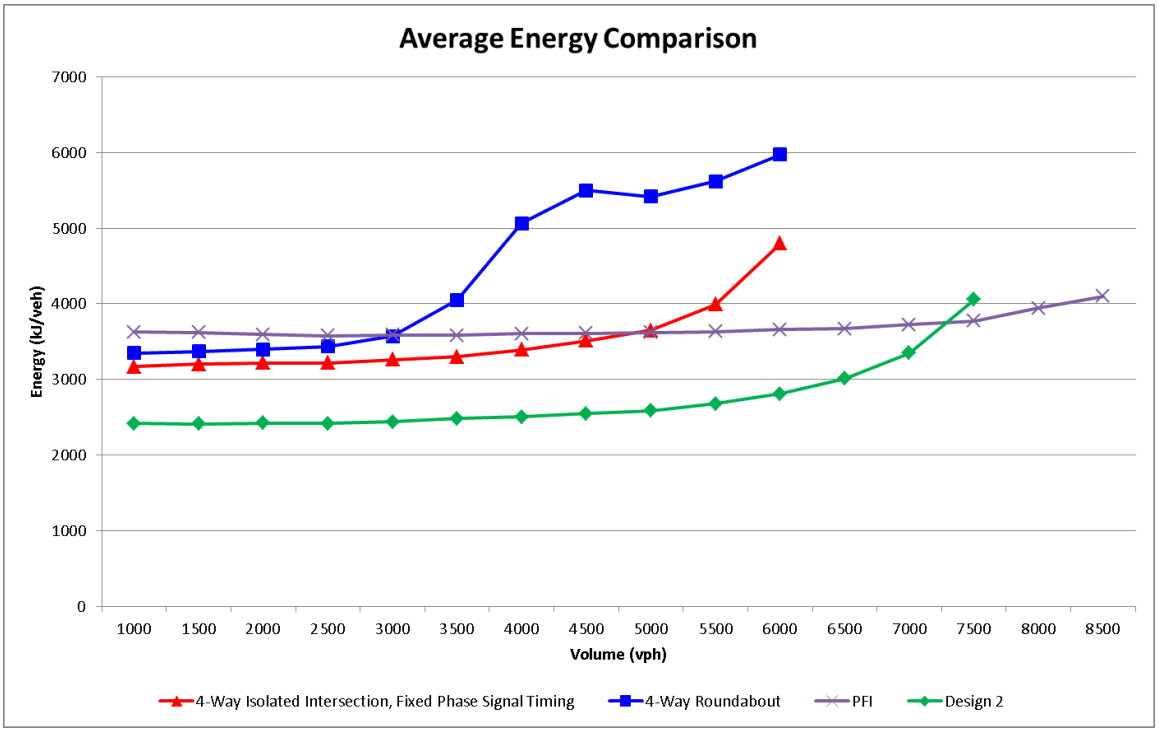

Figure 169: Average Energy Comparison of Truncated Square Intersection and Baseline Intersections

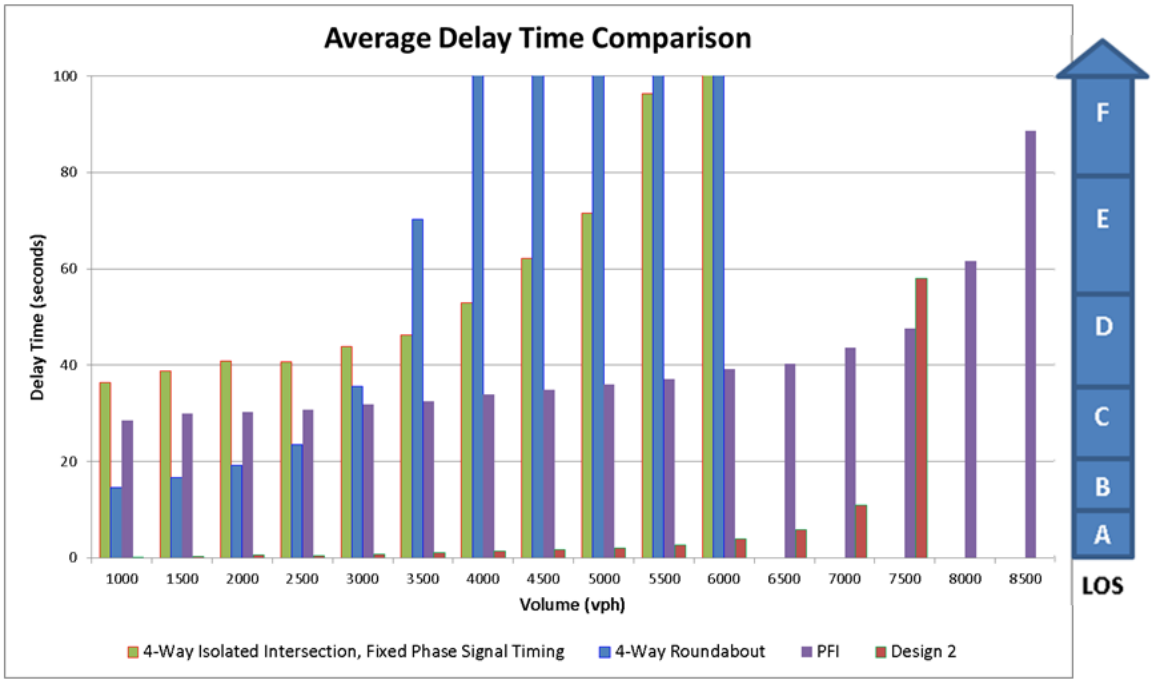

Figure 170: Average Delay Time Comparison with LOS shown at Right

6.2.4 Additional Sensitivity Analyses

6.2.4.1 Major/Minor Ratio Sensitivity Analysis

In addition to testing sensitivity to increases in traffic volume, several other sensitivity analyses can also be conducted. For example, a major/minor ratio sensitivity analysis examines how performance metrics such as travel time are affected by changes in the ratio of traffic on the main street to traffic on the cross-street. The major/minor ratio is important for appropriately setting fixed signal timing plans for signalized intersections. However, once a signal timing plan is set, traffic entering under major/minor ratios different than the design major/minor ratio may negatively impact the intersection's performance. A sensitivity analysis for the four intersections is shown in

Figure 171, where the sensitivity is absolute, and is measured relative to a major/minor ratio of 1.5 with the traffic volume set to 80% of each intersection's capacity (V/C = 0.8). The relative sensitivity values are shown in Table 66, where negative values indicate improvement with respect to the major/minor ratio of 1.5. The signal timing for the conventional 4-leg intersection was set to accommodate a major/minor ratio of 1.5, with all signal splits assumed to be known *a priori*.

The overall results indicate that the truncated square intersection is the least sensitive to changes in the major/minor ratio. The conventional 4-leg intersection is sensitive due to the signal timing being fixed. Round-about intersections generally perform better on average when the major/minor ratio is not equal to 1 due to traffic on the major street having less minor street traffic to negotiate in the round-about. The PFI performed slightly better when the major/minor ratio was reduced to 1; however, the travel time increased by over 9% when the ratio was increased from 1.5 to 2. Due to physical constraints in the design of PFI's, if the minor junctions on the major and minor streets are equidistant from the major junction, then the signal timing plan consists of two phases of equal length. If a PFI has two phases of equal length, the optimal major/minor ratio for the intersection is 1. Therefore, the PFI is more sensitive than other intersections to larger major/minor ratios. The truncated square intersection showed almost zero sensitivity to increases or decreases in the major/minor ratio. There are two reasons why the truncated square intersection is relatively insensitive to changes in the major/minor ratio. First, the truncated square intersection is an unsignalized intersection in which major and minor traffic are merging around a rounded square round-about. Fixed signal

intersections are sensitive to changes in the major/minor ratio. Although the round-about is also an unsignalized intersection, vehicles must yield to cross-street traffic that is already in the round-about. Second, in contrast to a round-about, vehicles in the truncated square intersection enter the rounded square round-about without having to yield to cross-street traffic. Therefore, in under-saturated conditions, the ratio of major to minor street traffic has little to no effect on the average travel time.

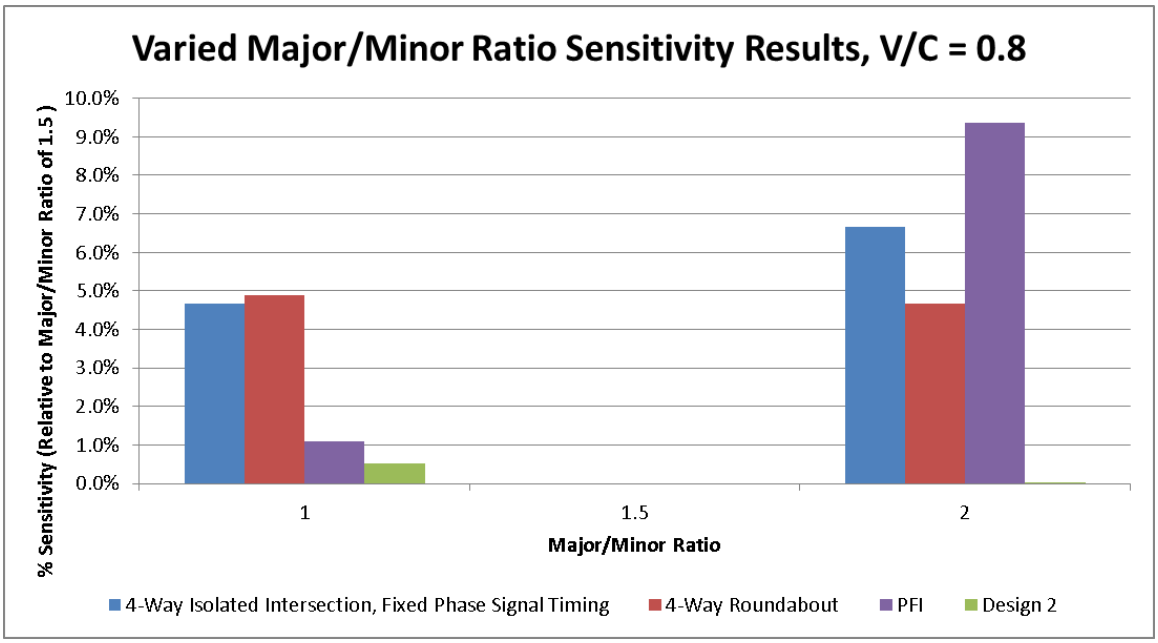

Figure 171: Major/Minor Ratio Sensitivity Results for Travel Time

Table 66: Major/Minor Ratio Sensitivity Results for Travel Time

		Traffic Intersection			
		Conventional 4-leg	Round-about	PFI	Truncated Square
Major/Minor Ratio	1	-5%	5%	-1 %	0.5%
	1.5	---	---	---	---
	2	7%	-5%	9%	0 %

6.2.4.2 Demand Profile Sensitivity Analysis

An additional sensitivity analysis was performed on the demand profile of traffic arriving at each of the intersections. A demand profile is a sequence that can modify an OD matrix to provide additional traffic during specified time intervals. For example, a sequence such as {25, 25, 25, 25} specifies that 25% of the traffic for a given OD pair should be released by the simulator during the first quarter of the simulation time period. A demand profile of {100} leaves the OD matrix unmodified. For the demand profile sensitivity test, the volume to capacity ratio was set to 0.5 for each intersection, and demand profile sequences with standard deviations of 1, 3, and 5 were tested relative to a demand profile of {100}. The specific demand profile sequences are shown in Table 67. The individual numbers in the demand profile sequences with non-zero standard deviation specify the percentage of the overall hourly volume for OD pairs for specific 5-minute intervals. The first number in the sequence specifies the first 5-minute interval during the hour-long simulation run. Although the overall hourly volume is set to 0.5 times the V/C ratio, the 5-minute interval volumes each have their own V/C ratio. The

individual sequence numbers were constrained to ensure that the 5-minute interval V/C ratios did not exceed 1. The signal splits for the conventional 4-leg intersection were set with the assumption that the ratios of traffic utilizing each signal phase were known perfectly *a priori*. The relative sensitivity results are shown in Figure 172 and Table 68 below.

The results indicate that the truncated square intersection is the least sensitive to variability in the arrival rates of vehicles. The conventional 4-leg intersection and the PFI had similar sensitivity to variations in the demand profile. Both of the intersections employed fixed signal timing where the cycle length was fixed. Ideally, if the entire demand profile sequence was known in advance, the cycle length could be adjusted for every 5-minute interval as necessary for the conventional 4-leg intersection. The round-about was the most sensitive to variations in the demand profile. In contrast, the truncated square intersection was the least sensitive to increases in the standard deviation of the demand profile. Although both the round-about and the truncated square intersection are unsignalized, the round-about is more sensitive to brief near-saturated conditions (V/C >= 0.9).

Table 67: Demand Profile Sequences used for Demand Profile Sensitivity Test

Standard Deviation of Demand Profile	Demand Profile Sequence
0	{100}
1	{8, 8, 7, 8, 9, 8, 10, 7, 9, 8, 10, 8}
3	{3, 9, 7, 12, 6, 14, 8, 7, 7, 11, 9, 7}
5	{3, 12, 4, 11, 13, 17, 5, 7, 3, 3, 14, 8}

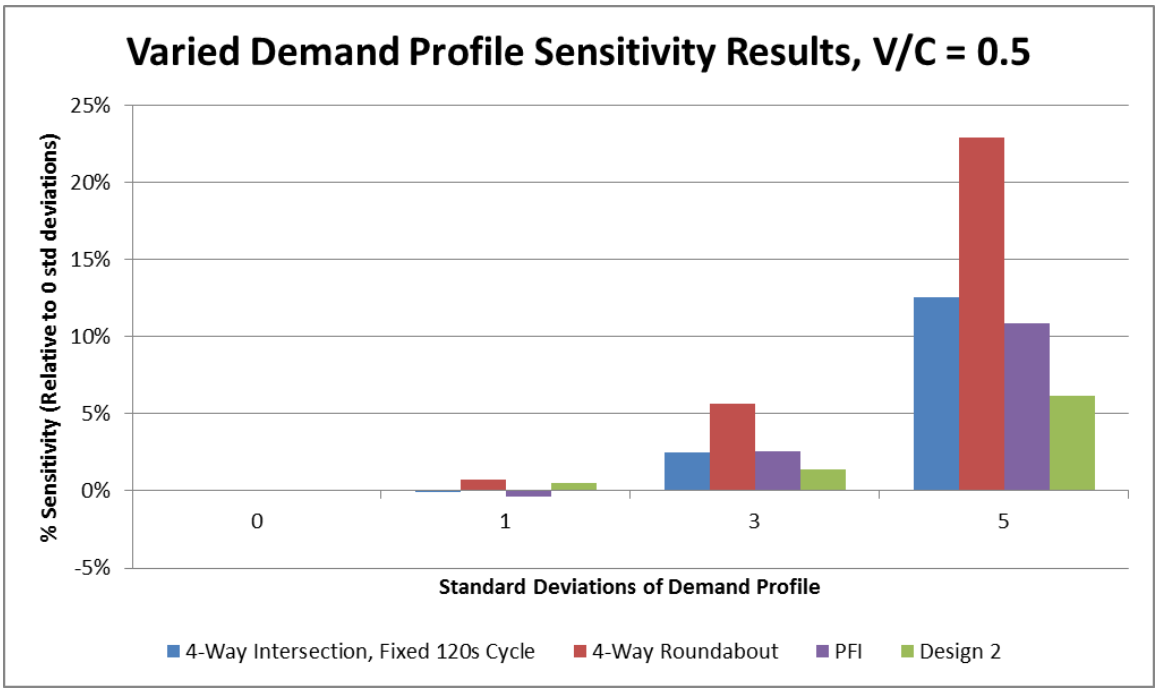

Figure 172: Demand Profile Sensitivity Results for Travel Time

Table 68: Demand Profile Sensitivity Results for Travel Time

		Traffic Intersection			
		Conventional 4-leg	Round-about	PFI	Truncated Square
Standard Deviations of Demand Profile	0	---	---	---	---
	1	0%	1%	0%	0%
	3	2%	6%	3%	1%
	5	13%	23%	11%	6%

6.2.4.3 Left-turn Percentage Sensitivity Analysis

The final sensitivity analysis was performed on sensitivity to increasing proportions of left-turns relative to through movement traffic. Left-turn sensitivity is

especially important in the context of UAIDs due to the presence of alternative left-turn treatments. Sensitivity was measured relative to a scenario of 20% left-turns, 70% through traffic, and 10% right-turns. The left-turn ratio was varied from 10% to 50% in increments of 10% with the right-turn ratio fixed at 10%. The volume for each intersection was fixed at 80% of the capacity of the intersection. In order to have a strong baseline, the cycle length for each left-turn ratio scenario for the conventional 4-leg intersection was determined using Webster's cycle length, where the OD matrix is assumed to be known. The unmodified equation for Webster's cycle length, C, is $C = (1.5 * L + 5)/(1 - CS/S)$, where L is the loss time in the cycle due to the duration of yellow and red signal phases, where CS is the sum of the critical lane volumes over every signal phase for the intersection, and where S is the saturation flow rate. After the Webster cycle length is determined, the signal splits are determined in the same manner as the signal splits for the HCM method ["HCM 2000", 2000]. The relative sensitivity results are shown in Figure 173 and Table 69 below. Negative values in the table indicate improvement in travel time.

The results indicate that the truncated square intersection is the least sensitive to relatively high ratios of left-turn traffic. Despite the handicap of being the only intersection to have direct access to the OD matrix data, the conventional 4-leg intersection with the Webster cycle was the most sensitive to left-turn traffic. The round-about and the PFI exhibited similar levels of sensitivity at high ratios of left-turn traffic. For example, the average travel time in a round-about and a PFI at a left-turn ratio of 40% is more than double the average travel time for a round-about and a PFI at a left-turn

ratio of 20%. For the round-about, increasing left-turn traffic corresponds to vehicles traveling further around the round-about and spending more time occupying space in the intersection. A round-about where left-turns are restricted has a higher capacity than a round-about where left-turns are permitted. In the case of the PFI, the inability to adjust the cycle length or cycle splits for a given intersection geometry leads to high sensitivity to imbalances in left-turn and through movement traffic. In contrast to the round-about, the truncated square intersection provides sufficient space in the round-about portion of the intersection to dampen the effect of increasing the left-turn ratio of traffic. Like the round-about, the capacity of a left-turn permissive truncated square intersection is lower than the capacity of a left-turn restricted truncated square intersection. However, the drop in capacity for the truncated square intersection due to permitting left-turns is less than the drop in capacity for the round-about due to permitting left-turns. In essence, the truncated square intersection is the least sensitive to high ratios of left-turn traffic due to the intersection having the highest left-turn capacity.

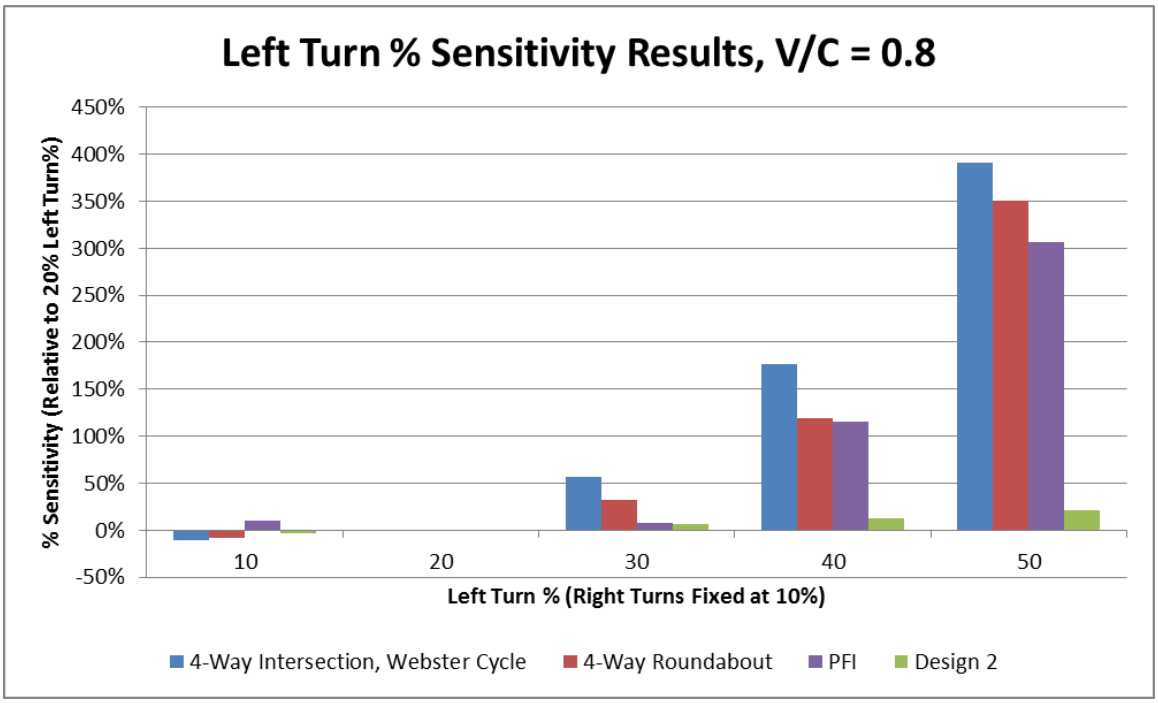

Figure 173: Left-turn Sensitivity Results for Travel Time

Table 69: Left-turn Sensitivity Results for Travel Time

		Traffic Intersection			
		Conventional 4-leg	Round-about	PFI	Truncated Square
Left-turn % (Right-turns fixed at 10%)	10%	-11%	-8%	10%	-4%
	20%	---	---	---	---
	30%	56%	33%	7%	6%
	40%	177%	119%	116%	13%
	50%	391%	351%	306%	22%

6.2.5 Range of Benefits

The range of benefits of using the truncated square intersection as opposed to a conventional 4-leg intersection, a round-about, or a PFI is shown in Table 70 below. The range of benefits of the truncated square intersection over the conventional 4-leg intersection and round-about is applicable to traffic volumes up to 6000 vehicles per hour. Since the capacity of the truncated square intersection is around 7500 vehicles per hour, the maximum travel time and energy benefits at traffic volumes greater than 6000 vehicles per hour are higher than the values shown in the table. Accordingly, the range of benefits shown in Table 70 represent conservative estimates of the benefits of using the truncated square intersection, and treat additional capacity as a separate benefit. The range of benefits of the truncated square intersection over the PFI is applicable to traffic volumes up to 7500 vehicles per hour. Since the capacity of the PFI, 8500 vehicles per hour, is greater than the capacity of the truncated square intersection, the lower end of the range of benefits is lower for traffic volumes greater than 7500 vehicles per hour. Nevertheless, the truncated square intersection provides substantial benefits over a wide range of traffic volumes over several existing intersection designs.

Table 70: Range of Benefits for Truncated Square Intersection over Baseline Intersections

% Savings of Proposed Design vs…:	Travel Time	Energy	CO_2	CO	HC	NO_x	$PM_{2.5}$
Conventional	35% - 75%	24% - 41%	24% - 41%	36% - 48%	40% - 52%	19% - 25%	31% - 63%
Round-About	21% - 95%	28% - 53%	28% - 53%	46% - 54%	41% - 64%	30% - 36%	39% - 66%
Parallel Flow Intersection	-25% - 33%	-8% - 33%	-8% - 33%	11% - 60%	2% - 49%	-2% - 40%	24% - 74%

6.2.6 Benefit-Cost Analysis

A benefit-cost analysis (BCA) was conducted to determine the economic feasibility of constructing, maintaining, and operating a truncated square intersection in comparison to a conventional 4-leg intersection. There are several methods of conducting BCAs. The BCA presented is specific to transportation and is based on the New York State Department of Transportation's BCA method, as described by the Federal Highway Administration [Gordon, 2012]. The financial cost inputs for the BCA are shown in Tables 71-73 below. The roadway cost per lane-mile is shown in Table 71. The roadway costs for the truncated square intersection are highlighted in yellow and the roadway costs for the conventional 4-leg intersection are highlighted in orange. The maintenance rates for signal controllers are shown in Table 72, and the road maintenance cost per lane per mile is shown in Table 73.

Tables 71-73: Financial Cost Inputs of BCA

Roadway Cost Per Lane-Mile	Rural	Urban
1 lane each direction	$ 2,946,564.50	$ 4,207,024.00
2 lanes each direction	$ 2,351,336.50	$ 2,969,312.50
3 lanes each direction	$ 1,998,314.67	$ 2,423,816.00

Maintenance Rates (RITA-2005)	
Maintenance Item	*Rates of Failure (per year)*
Controller Failure	0.1
Controller Upgrade	0.1
Signal Timing Adjusting	0.25

Road Maintenance	
Maintenance cost per lane per mile	$ 4,381.00

Proposed
Conventional

The construction, operational and maintenance costs are shown in Table 74. Although the truncated square intersection is slightly less expensive to construct than a conventional intersection, the necessity of including pedestrian overpasses for the truncated square intersection makes the total road construction cost slightly higher for the proposed intersection. The operational and maintenance costs are lower for the truncated square intersection due to less roadway area and the absence of traffic signals and pedestrian pushbuttons.

Table 74: Financial Cost of Conventional 4-leg Intersection and a Truncated Square Intersection

Cost Categories	Subcategories	Conventional	Proposed
Road	***New Road Construction Cost***	$ 7,845,804.63	$ 7,198,333.33
	Pedestrian Overpass Cost	$ -	$ 1,500,000.00
	Overpass unit cost		$ 750,000.00
	Total Road Construction Cost	$ 7,845,804.63	$ 8,698,333.33
	Annual Road Maintenance Cost	$ 14,181.14	$ 10,620.61
Traffic Lights	***Purchase + Installation***	$ 200,000.00	$ -
	Accessible Pedestrian Signals	$ 400.00	
	Pedestrian Pushbuttons	$ 400.00	
	Countdown Pedestrian Signals	$ 300.00	
	Annual Electricity Costs	$ 1,400.00	$ -
	Average Annual Maintenance	$ 4,400.00	$ -
	Controller Replacement	$ 26,500.00	
	Controller Upgrade	$ 10,000.00	
	Signal Timing Adjusting	$ 3,000.00	
	Traffic Light Total First Year Cost	$ 205,800.00	$ -
	Total Annual Operational Costs	$ 1,400.00	$ -
	Total Maintenance Costs	$ 18,581.14	$ 10,620.61

The inputs for the BCA are presented in Table 75. The system operational life is a conservative estimate of the life of streets constructed with asphalt; (streets constructed with concrete have a higher system operational life). Using microsimulation, values relating to mobility and energy were generated for Table 76. The equations for the BCA are shown in Table 77. The equation for the network operational life monetary benefit assumes a linear scaling of benefits between an individual isolated intersection and a 10 by 10 grid of intersections. In reality, a system of coordinated conventional intersections might produce a lower average travel time than a single isolated conventional

intersection. The BCA is shown in Table 78. The annualized monetary benefit of constructing, maintaining, and operating the truncated square intersection instead of the conventional 4-leg intersection is over 11 million dollars. The operational life monetary benefit of over 157 million dollars is a conservative value based on a relatively low system operational life. The benefit-to-cost ratio is 12.5. The final value of roughly 15.7 billion dollars represents the monetary benefit for a small city. The New York State Department of Transportation BCA is based on construction costs, maintenance costs, operational costs, the time value of money, the cost of crashes, and the cost of fuel; however the BCA does not include environmental criteria. A quantification of the monetary benefit of reducing emissions in a concentrated urban environment would lead to an even higher benefit-to-cost ratio.

Table 75: Benefit-Cost Analysis Inputs

Inputs	*Abb.*	
Interest Rate (decimal)	I	0.0608
System Operational Life (years)	NL	15
Daily Traffic Volume (veh/day)	DTV	50000
Private Vehicle % of Volume	PVV	0.95
Commercial Vehicle % of Volume	CVV	0.025
Freight Vehicle % of Volume	FVV	0.025
Average Private Vehicle Occupancy Rate	K_1	1.55
Average Commercial Vehicle Occupancy Rate	K_2	1.11
Average Freight Vehicle Occupancy Rate	K_3	1.11
Private Vehicle Occupant Time Rate ($ / vehicle occupant hour)	H_1	$ 17.02
Commercial Vehicle Occupant Time Rate ($ / vehicle occupant hour)	H_2	$ 27.49
Freight Vehicle Occupant Time Rate ($ / vehicle occupant hour)	H_3	$ 30.81
Annual average conventional intersection crash rate	CCRA	4.71
Expected annual average proposed intersection crash rate	PCRA	0.471
$ Cost per crash	H_4	$ 45,585.00
Cost of gasoline ($ / gallon)	H_5C	$ 3.93
Cost of diesel ($ / gallon)	H_5CF	$ 4.02
Density of gasoline (g/cm^3)		0.7
Density of diesel (g/cm^3)		0.9

Table 76: Simulated and Projected Values for Benefit-Cost Analysis

Simulated & Projected Values (based on 5000 vph)	*Abb.*	
Conventional Average Private Vehicle Travel Time (seconds)		142.5448
Conventional Average Commercial Vehicle Travel Time (seconds)		152.1183
Conventional Average Freight Vehicle Travel Time (seconds)		152.1183
Conventional Annual Private VHT	CAPVHT	686491.97
Conventional Annual Commercial VHT	CACVHT	19278.8768
Conventional Annual Freight VHT	CAFVHT	19278.8768
Proposed Average Private Vehicle Travel Time (seconds)		68.7539
Proposed Average Commercial Vehicle Travel Time (seconds)		67.7404
Proposed Average Freight Vehicle Travel Time (seconds)		67.7404
Proposed Annual Private VHT	PAPVHT	331117.026
Proposed Annual Commercial VHT	PACVHT	8585.15852
Proposed Annual Freight VHT	PAFVHT	8585.15852
Conventional Average Fuel Consumption (grams)		94.9516516
Conventional Average Diesel Consumption (grams)		312.742084
Proposed Average Fuel Consumption (grams)		53.3275448
Proposed Average Diesel Consumption (grams)		256.786426
Conventional Average Fuel Consumption (gallons)	CAFC	0.03583747
Conventional Average Diesel Consumption (gallons)	CADC	0.09180718
Proposed Average Fuel Consumption (gallons)	PAFC	0.02012734
Proposed Average Diesel Consumption (gallons)	PADC	0.07538108

Table 77: Benefit-Cost Analysis Equations

Category	*Abb.*	*Equation*
Design Cost + Construction Costs	PDC	$Design\ Costs\ +\ Construction\ Costs$
Capital Recovery Factor	CRF	$\frac{I * (1+I)^{NL}}{(1+I)^{NL} - 1}$
Uniform Annual Equivalent Investment Cost	REI	$PDC * CRF$
Annualized Life-Cycle Cost	LCC	$REI + AOC + AMC$
Private Vehicle Travel Time* (Person hours)	LPP	$K_1 * (C\ or\ P)APVHT$
Commercial Vehicle Travel Time (Person hours)	LPT	$K_2 * (C\ or\ P)ACVHT$
Freight Vehicle Travel Time (Person hours)	LPG	$K_3 * (C\ or\ P)AFVHT$
Annual Private Vehicle Gasoline (gallons/year)	APVG	$DTV * PVV * (C\ or\ P)AFC * 365$
Annual Commercial/Freight Vehicle Diesel (gallons/year)	ACFVD	$DTV * (CVV + FVV) * (C\ or\ P)ADC * 365$
Private vehicle occupant system time $ value	PVOSD	$H_1 * LPP$
Commercial vehicle occupant system time $ value	CVOSD	$H_2 * LPT$
Freight vehicle occupant system time $ value	GID	$H_3 * LPG$
Annual Cost of Crashes	CC	$H_4 * (C\ or\ P)CRA$
Annual Cost of Gasoline	CG	$H_{5C} * APVG$
Annual Cost of Diesel (all commercial & freight)	CD	$H_{5CF} * ACFVD$
Annualized Monetary Performance	MP	$PVOSD + CVOSD + GID + CC + CG + CD$
Annualized Monetary Benefit (Proposed over Conventional)	MB	$MP(C) - MP(P)$
BENEFIT/COST RATIO	B/C	MB/LCC
Operational Life Monetary Benefit	OLMB	$(MB\ - LCC) * NL$
Network Operational Life Monetary Benefit (10 x 10 network)		$OLMB * 100$

Table 78: Benefit-Cost Analysis

Category	*Abb.*	*Conventional*	*Proposed*
Design Cost + Construction Costs	PDC	$ 7,845,804.63	$ 8,698,333.33
Capital Recovery Factor	CRF	0.103501642	0.103501642
Uniform Annual Equivalent Investment Cost	REI	$ 812,053.66	$ 900,291.78
Annualized Life-Cycle Cost	LCC	$ 832,034.80	$ 910,912.39
Private Vehicle Travel Time* (Person hours)	LPP	1064062.553	513231.3902
Commercial Vehicle Travel Time (Person hours)	LPT	21399.55327	9529.525962
Freight Vehicle Travel Time (Person hours)	LPG	21399.55327	9529.525962
Annual Private Vehicle Gasoline (gallons/year)		621332.0704	348957.7405
Annual Commercial/Freight Vehicle Diesel (gallons/year)		83774.04828	68785.23728
Private vehicle occupant system time $ value	PVOSD	$ 18,110,344.66	$ 8,735,198.26
Commercial vehicle occupant system time $ value	CVOSD	$ 588,273.72	$ 261,966.67
Freight vehicle occupant system time $ value	GID	$ 659,320.24	$ 293,604.69
Annual Cost of Crashes	CC	$ 214,705.35	$ 21,470.54
Annual Cost of Gasoline	CG	$ 2,441,213.70	$ 1,371,054.96
Annual Cost of Diesel (all commercial & freight)	CD	$ 336,436.58	$ 276,241.51
Annualized Monetary Performance	MP	$ 22,350,294.25	$ 10,959,536.64
Annualized Monetary Benefit (Proposed over Conventional)	MB		$ 11,390,757.61
BENEFIT/COST RATIO	B/C		12.50477848
Operational Life Monetary Benefit			$ 157,197,678.34
Network Operational Life Monetary Benefit (10 x 10 network)			$ 15,719,767,834.04

6.2.7 Design Parameter Optimization for Truncated Square Intersection

The truncated square intersection design can be expressed using several design parameters. The proposed truncated square intersection design may be 1) used as an isolated arterial intersection, 2) used as an at-grade freeway interchange, or 3) tiled into an arterial traffic network. In each of these cases, parameters may be modified to best fit the topography and application type. As shown in Figure 174, there are four geometric design parameters which may be modified: 1) *a*, the square round-about diameter, 2) *b*,

the bi-directional link length, 3) *c*, the square round-about turning radius, and 4) *d*, the octagon round-about turning radius. Additional parameters include: 1) specification of the speed limit, 2) the number of lanes approaching the intersection, and 3) the number of lanes within the round-about.

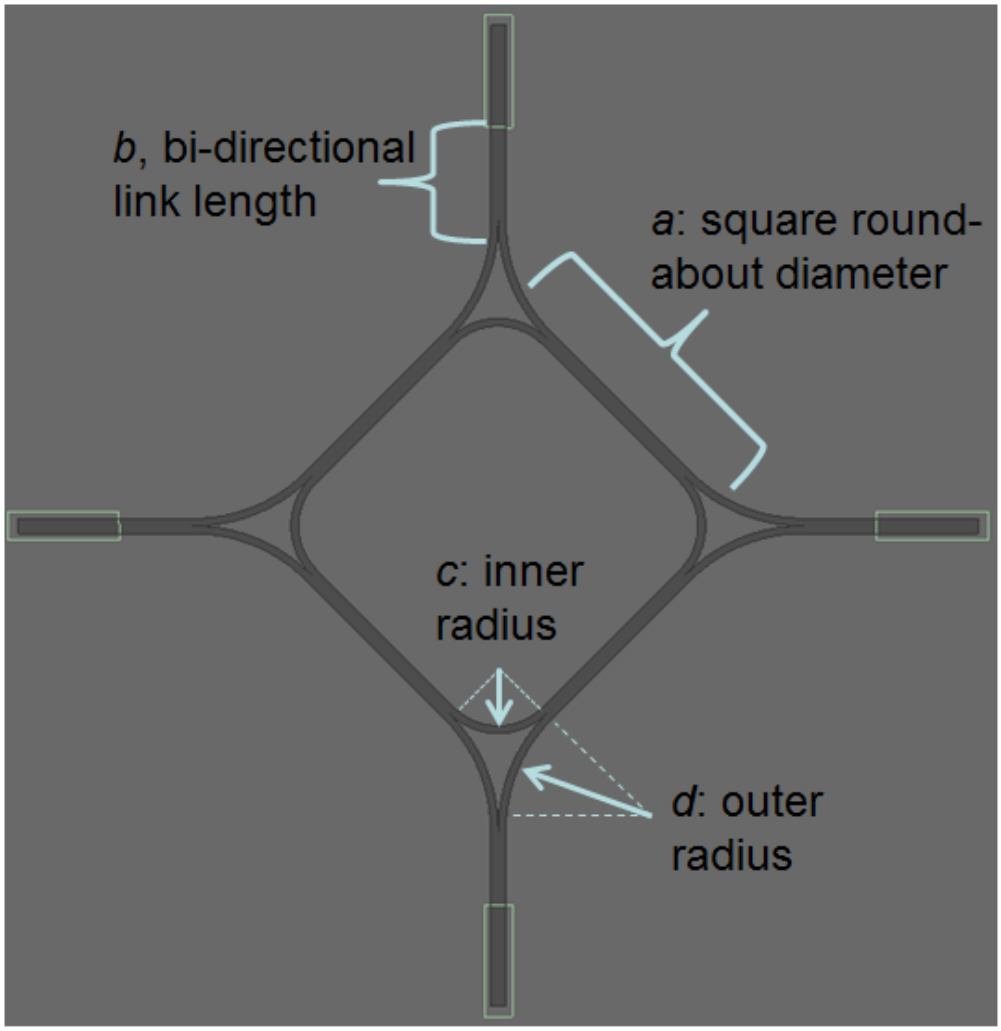

Figure 174: Geometric Design Parameters for Truncated Square Intersection

Regarding geometric parameter selection, *a*, is selected to provide sufficient time for drivers to make lane changes, and *c* and *d* are selected to permit vehicles to travel at

the speed limit during the turns. The larger the speed limit, the larger *a*, *c*, and *d* must be in order to permit vehicles to travel near the speed limit throughout the network. Therefore, due to the difference in speed limits, the geometric parameters selected for a freeway interchange would be substantially larger than the geometric parameters selected for an isolated arterial intersection, or an urban traffic network. In the case of an urban city traffic network, the selection of *b* determines spacing between intersections and is correlated to the relative travel time improvement between the proposed network and the conventional grid iron network. Larger *b* leads to better relative performance with the tradeoff of larger city block sizes. The capacity of the network is related to the selection of the number of lanes, the speed limit, and *a*. Increasing the number of lanes increases the capacity of the network at the expense of lowering the land utilization rate.

The truncated square intersection has a higher free-flow travel time than a conventional 4-leg intersection. The percent increase in free-flow travel time can be adjusted by changing *a* and *b*. The relative increase in free flow travel time is shown in Figure 175 below. In order to maximize the benefits of improved travel time, it is desirable for the percent increase in free-flow travel time to be minimized. In the worst case scenario, selecting $b = 0$ leads to the free-flow travel time being 41% higher for the truncated square intersection than the conventional 4-leg intersection. Coincidentally, when *b* is set to 0, the truncated square network becomes a rounded rectified square network, as described in section 6.1.5.1. As *b* is increased to infinity, the difference in free-flow travel times approaches 0. In the context of urban design, increasing *b* corresponds to increasing the size of city blocks. Therefore, although a large *b* is

desirable in terms of percent savings of free-flow travel time, *b* is in reality constrained by the maximum permissible city block size. Decreasing *a* also reduces the percent difference in free-flow travel times. Furthermore, reducing *a* also reduces the capacity of the intersection and the time that drivers have to make lane changes. Therefore, the minimum acceptable *a* is determined by the minimum acceptable capacity, and the minimum acceptable time allotted for drivers to make lane changes. Deploying connected and automated vehicles would lead to a reduction in the minimum acceptable *a* due to lane changes requiring less time and vehicles being able to travel closer together.

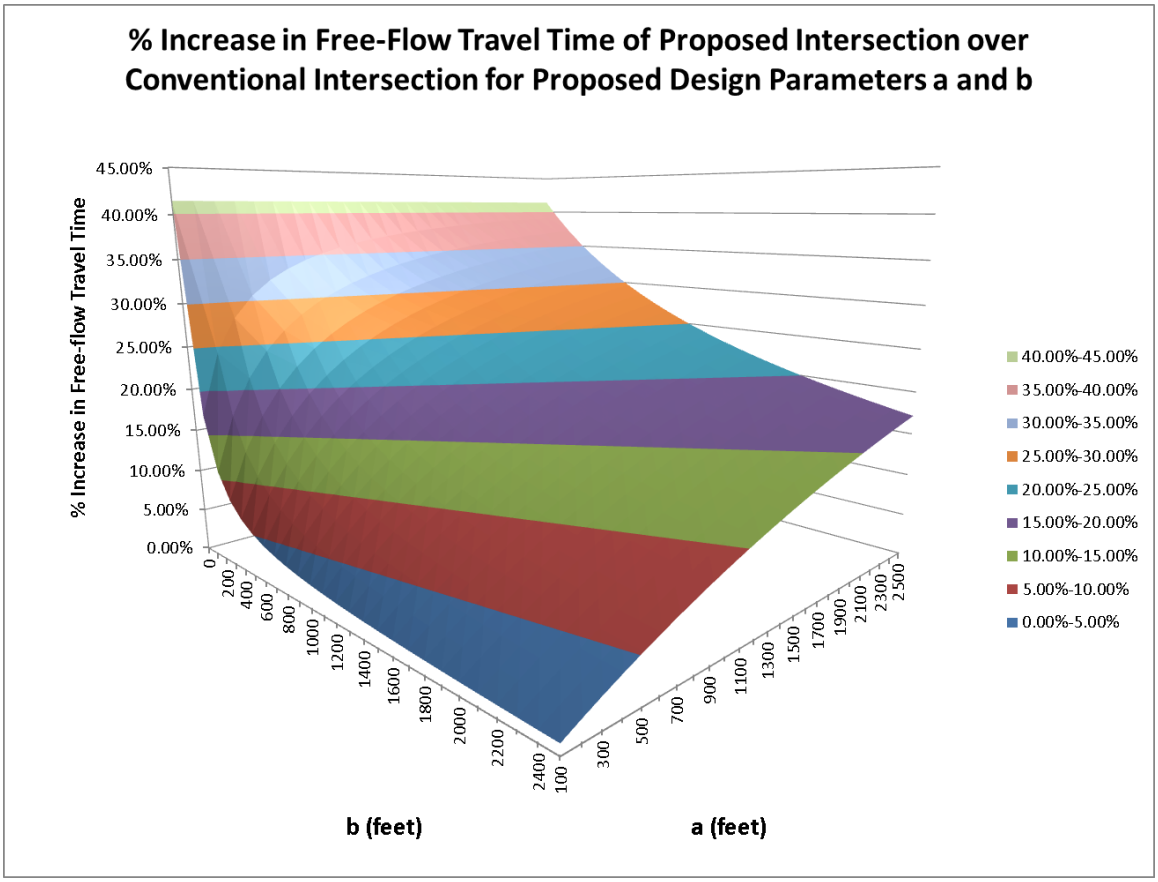

Figure 175: Percent Increase in Free-flow Travel Time of a Truncated Square Intersection defined by (*a*, *b*) over a Conventional 4-leg Intersection

The percent increase in free-flow travel time of one intersection over another is identical to the percent increase in route distance. However, route distance is not necessarily correlated to travel time. Furthermore, in terms of mobility, the free-flow travel time is only one component of the overall travel time. Travel time is defined as the sum of free-flow travel time and delay time. The average travel time percent savings of the truncated square intersection over a conventional 4-leg intersection is shown in Figure 176 below. The estimation is bases on the conventional 4-leg intersection operating at its capacity with an average delay time of 80 seconds. The equivalent sized truncated square intersection is assumed to have an average delay of 2.5 seconds. It is important to note that the average delay of a conventional 4-leg intersection does not vary with design parameters *a* and *b*. Similarly, the average delay of a truncated square intersection is assumed to not vary significantly with the design parameters *a* and *b*. Across the tested range of values for *a* and *b*, the average travel time percent savings varied from between 26% and 93%. The average travel time percent savings are relatively low when *a* and *b* are large. Accordingly, the average travel time percent savings are relatively high when *a* and *b* are small. Although using a large *b* is advantageous for minimizing the percent increase in route distance and free-flow travel time, it is a small *b* that provides the optimal savings in travel time. In fact, the results suggest that a *b* of 0 provides the optimal percent travel time savings. The actual travel time savings are due to the rounded-square round-about feature, as parameterized by *a*, of the truncated square intersection design. The bi-directional links connecting the square round-abouts, as parameterized by *b*, actually lower the travel time savings because the ratio of free-flow

travel time to delay time is larger for larger *b*. Despite the fact that a *b* of 0 leads to the maximum increase in free-flow travel time, the delay time is the more significant term at smaller values of *b*. Recall that when *b* is 0 the truncated square intersection becomes a rounded rectified square intersection. As a result, the city block size is uniform and is determined by the selection of *a*. The value of *a* is chosen to balance average travel time savings and the overall capacity of the intersection. In summary, because the absolute delay time savings of the truncated square intersection over the conventional 4-leg intersection are essentially fixed over different selections of *a* and *b*, the average travel time percent savings are dictated by minimizing the absolute amount of free-flow travel time of the truncated square intersection (Figure 177) and the conventional 4-leg intersection as opposed to minimizing the relative percent increase in free-flow travel time (Figure 175) of the truncated square intersection over a conventional 4-leg intersection.

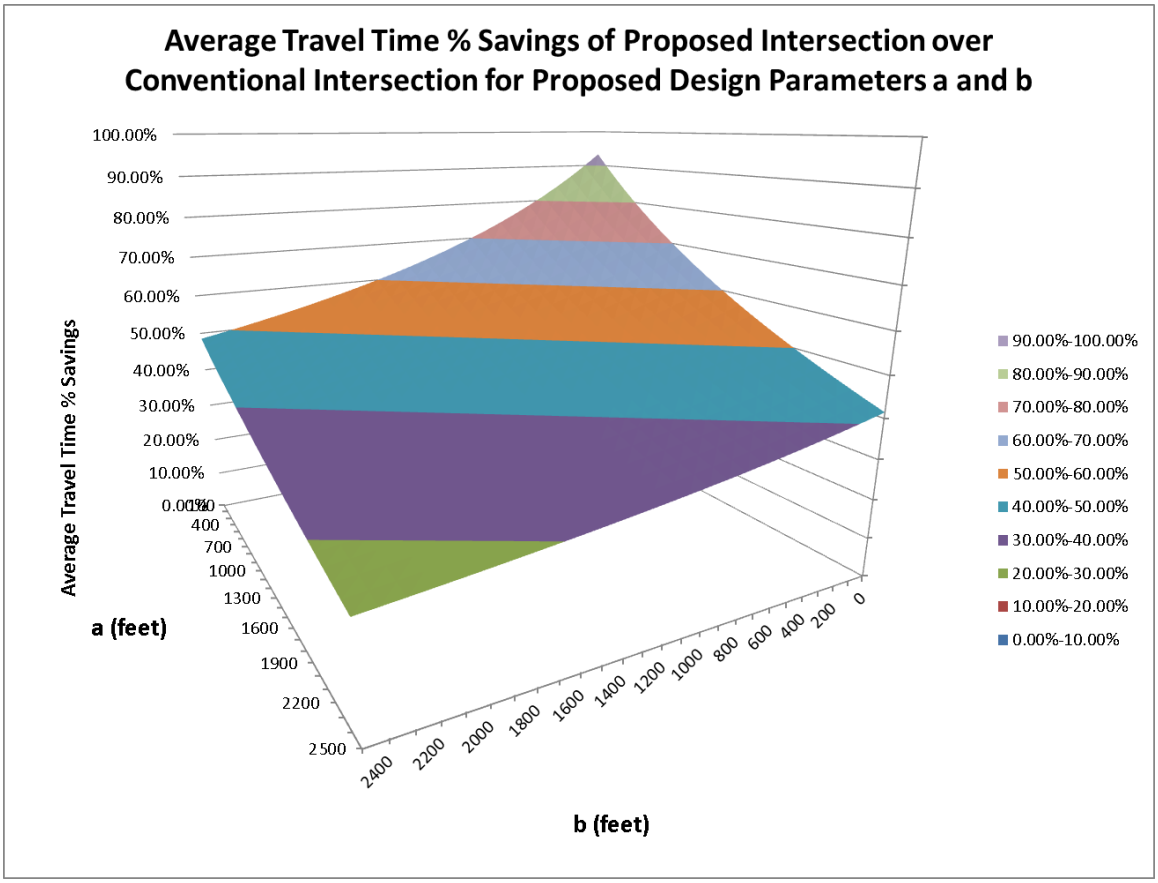

Figure 176: Average Travel Time Percent Savings of a Truncated Square Intersection defined by (*a*, *b*) over a Conventional 4-leg Intersection

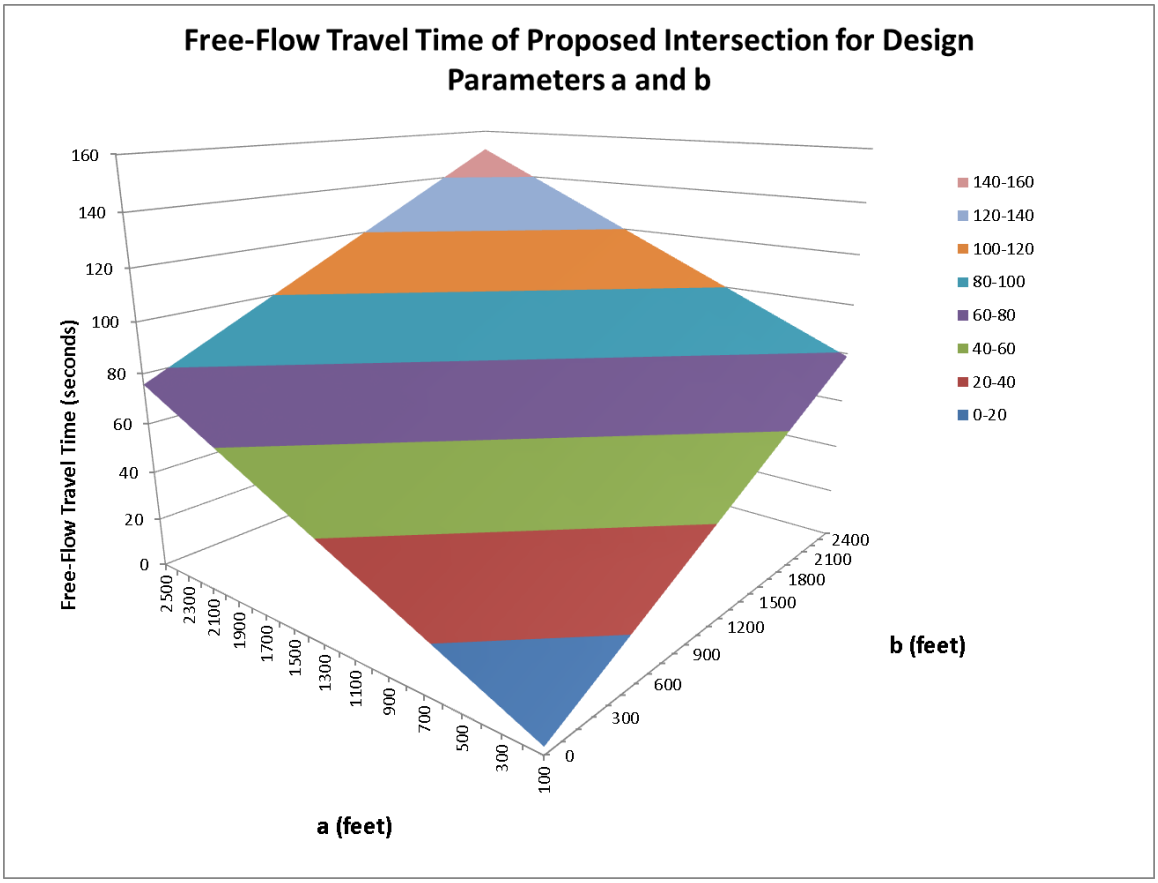

Figure 177: Absolute Free-flow Travel Time of a Truncated Square Intersection defined by (*a*, *b*)

7. Conclusions & Future Work

As stated in the introduction, vehicular transportation systems may be viewed as containing three layers, namely, vehicles, infrastructure, and an interaction/cooperation layer. The advent of CV technology enables an overhaul of the interaction/cooperation layer via direct point-to-point wireless communications. In addition, CV technology enables the transformation of outdated transportation systems into intelligent transportation systems (ITS). The three main chapters in the dissertation, (chapters 4-6), explored 3 different approaches to creating a more intelligent transportation system. The solutions were proposed in the context of both arterial traffic systems and freeway traffic systems, and in terms of either "enhancement" or "modification" of infrastructure or vehicles. Recall that the term "enhanced" is used to indicate the addition of equipment to accommodate new capabilities, and that the term "modified" is used to denote alternative implementations that achieve similar or identical functionality. In chapter 4, the approach taken was to enhance the vehicle layer by incorporating communication into the interaction/cooperation layer. The application of Eco-CACC was demonstrated in the context of freeway transportation systems. In chapter 5, the approach taken was to enhance infrastructure and vehicles for the purpose of communicating information to enable the infrastructure to better serve vehicles in the context of arterial traffic signal control. The application of CV adaptive signal control demonstrated the benefits of enabling the interaction/cooperation layer to include vehicle-to-infrastructure (V2I) communications. In chapter 6, the approach taken was to modify infrastructure to

provide an improved foundation for traffic management of existing vehicles as well as the intelligent vehicles of tomorrow. The remainder of this chapter is organized as follows. First, the key findings, conclusions, and opportunities for future work specific to chapters 4, 5, and 6 are presented in sequential order for each of the traffic simulation networks. Finally, based on the totality of information presented in the dissertation, additional avenues for future work are presented.

In chapter 4, a series of four sensitivity analyses were conducted for network 1 concerning traffic volume, penetration rate of Eco-CACC technology, triggering distance, and intra-platoon spacing. For the traffic volume sensitivity analysis, a maximum savings of over 70% and 30% were achieved in travel time and energy, respectively. In addition, the results indicate that the capacity is improved by at least two-thirds. Accordingly, the first conclusion that may be drawn is that platooning increases traffic density and roadway capacity. Furthermore, the simultaneous increase in traffic density and vehicle cooperation leads to the maintenance of free-flow traffic at higher volumes than are normally possible for today's freeways.

For the penetration rate of Eco-CACC technology, the higher the penetration rate, the greater the benefits in capacity improvement. In fact, the benefits in capacity improvement, travel time, and energy all appear to be positively correlated with the penetration rate. Penetration rates less than or equal to 10% have little benefit due to the sparsity of platoons. Across the range of tested traffic volumes, penetration rates greater than or equal to 60% have similar performance to a 100% penetration rate scenario.

For the triggering distance sensitivity analysis, increasing the triggering distance results in a benefit of less than 3% for travel time and energy with respect to the baseline. Although increasing the triggering distance influences how early a vehicle joins a platoon, the overall length of platoons, and to a small degree the number of vehicles involved in platooning behavior, increasing the triggering distance does not have a significant impact on overall network benefits.

For the intra-platoon spacing sensitivity analysis, increasing the intra-platoon spacing results in a penalty of less than 1% for travel time and 2% for energy with respect to the baseline. Increasing intra-platoon spacing leads to increases in average platoon size and length (the number of vehicles within a platoon and the distance between the leader and the tail of the platoon, respectively). However, increasing intra-platoon spacing also lowers the total maximum capacity. Therefore, there is a slight trade-off between overall capacity improvement and level of safety.

For network 2, the hypothetical freeway segment with on/off ramps and a dedicated lane, a traffic volume sensitivity analysis was conducted. The benefits for volumes less than 6600 vph were 0-2% for travel time and 0-4% for energy. However, there is a significant improvement at traffic volumes greater than 6600 vph, around 28% for travel time and 9% for energy at 7800 vph. At higher traffic volumes, the dedicated lane energy savings increased to a maximum of 14%. In addition, the indirect benefit of non-dedicated lane energy savings reached roughly 4% at the highest volume. In conclusion, even the presence of a single dedicated lane leads to a significant increase in

overall network capacity. Furthermore, vehicles may maximize their energy savings by choosing a dedicated lane where platooning is permitted as opposed to a lane where platooning is prohibited. However, vehicles which remain outside of the dedicated lane still benefit indirectly in terms of travel time and energy.

For network 3, the SR-91 E freeway with a dedicated lane, a traffic volume sensitivity analysis was conducted. When the vehicle spacing parameter was set to 5 meters, the benefits ranged from 0-42% for travel time and 0-19% for energy. Likewise, when the vehicle spacing parameter was set to 15 meters, the benefits ranged from 0-24% for travel time and 0-13% for energy. In conclusion, vehicle spacing has a significant impact on overall network benefits due to its inverse relationship with overall network capacity. Regardless of vehicle spacing, the presence of only a single dedicated lane still led to a significant increase in overall network capacity. The dedicated lane was 3-12% more energy efficient than the non-dedicated lanes, and 3-26% more energy efficient relative to the average baseline scenario lane. Accordingly, vehicles may maximize their energy savings by choosing the dedicated "eco-lane." Vehicles which remain outside of the dedicated lane still benefit indirectly in terms of travel time and energy.

In terms of future work, additional testing may be conducted at even higher traffic volumes to more adequately quantify the increase in network capacity resulting from Eco-CACC. In particular, the relationship between vehicle spacing (or headway), to network capacity is of interest. For the SR-91 E freeway network, additional testing at

higher volumes is necessary in order to quantify the differences in capacity between the 5 and 15 meter inter-platoon spacing parameter values.

Although a penetration rate sensitivity analysis was conducted for network #1, a penetration rate sensitivity analysis can also be conducted for both networks #2 and #3. Although triggering distance does not have a significant impact on overall network benefits, triggering distance in conjunction with various controller parameters and designs (see Appendix A) may have a moderate impact on individual vehicle performance. An additional point of exploration might examine whether it is energy efficient to encourage Eco-CACC equipped vehicles to bypass unequipped vehicles in order to join a platoon. If it is energy efficient, the question arises concerning how aggressively an equipped vehicle can seek out platoons while attempting to save energy. The addition of the "bypass" protocol would mitigate the need for a specific triggering distance.

In the context of networks 2 and 3, the single dedicated lane proved to be the most energy efficient. Although the dedicated lane is the most energy efficient, it is not feasible for all drivers (under a 100% penetration rate scenario) to attempt to use the same lane at once. Additional research may be conducted to determine whether natural human decision making is sufficient in preventing overcrowding of the dedicated lane, and whether an optimal lane selection strategy could substantially improve upon human decision lane choices.

In light of the substantial benefits afforded by a single dedicated lane, a natural avenue of future research is to consider whether additional dedicated lanes lead to even more dramatic increases in roadway capacity. The following three paragraphs explore how the number of dedicated lanes in each of the three simulation networks affected the results.

Recall that network 1, the first hypothetical freeway segment, consisted of 2-3 lanes, including a lane-drop, and that the two left-most lanes were set as dedicated lanes. Due to the selection of two dedicated lanes, a merging protocol specifically designed to work with a single lane-drop was used in conjunction with the core Eco-CACC algorithm. The combination of the two algorithms proved to be highly effective, leading to benefits in travel time and energy over 70% and 30%, respectively. Furthermore, the initial baseline capacity of approximately 3600 vph was increased by two-thirds to 6000 vph, with additional capacity benefits likely at higher volumes (> 6000 vph). In fact, in order to fully assess the increase in capacity, platoons of vehicles will need to be inserted into the network, rather than inserting individual vehicles, to achieve the desired hourly insertion rate. The selection of only a single dedicated lane for network 1 would have removed the necessity of applying the merging protocol. Future work involves testing the hypothesis that a single dedicated lane would be worse than two dedicated lanes with the merging protocol.

Network 2, the second hypothetical freeway segment, consisted of 4 lanes, with the left-most lane set as a dedicated lane. Due to the presence of off-ramps, a platoon

splitting protocol was implemented in addition to the Eco-CACC core. Another hypothesis which can be tested in future work is that the presence of two or more dedicated lanes is better than a single dedicated lane. Including more than one dedicated lane for network 2 will involve another layer of interaction and cooperation, namely inter-platoon interaction and cooperation. The developed protocols described in section 4.1.3 involve intra-platoon cooperation, in which vehicles within a single platoon accommodate the merging or departure of a vehicle into or from a platoon. Inter-platoon cooperation may involve two or more platoons splitting in order to permit a vehicle to reach its desired exit.

Network 3, the SR-91 E freeway, consisted of 4-6 mainstream lanes, with the left-most lane set as a dedicated lane. Once again, the platoon splitting protocol was included along with the Eco-CACC core. Due to the absence of lane-drops, the inclusion of the platoon merging protocol was unnecessary. Based on the results for all three networks, it appears that the ratio of the number of dedicated lanes to total lanes is a significant factor in determining overall benefits. Accordingly, future work will consider conducting a sensitivity analysis on the number of dedicated Eco-CACC lanes within the SR-91 E freeway network. For each number of dedicated lanes, different auxiliary technologies may need to be applied to work in conjunction with Eco-CACC based on network geometry. The inclusion of all lanes as dedicated lanes will necessitate the creation of merging protocol designed around on-ramps. Due to the differences in relative speeds and the finite space of an on-ramp, the new merging protocol will likely be more complex than the developed lane-drop merging protocol. One hypothesis is that the ratio of the

number of dedicated eco-lanes to the total number of lanes may be one of the determining factors in terms of maximum possible benefits.

Lastly, the influence of acceleration and deceleration constraints on Eco-CACC performance was examined with the aid of an acceleration test and a deceleration test on network 3, the SR-91 E freeway. The results indicated that the selection of the acceleration and deceleration constraint values does not significantly impact Eco-CACC performance averaged over several miles of simulated real-world freeway. Although selecting lower acceleration constraint values temporarily decreased the energy rate, relative to a higher acceleration constraint value, the accompanying increase in the time duration of the gap-closing maneuver left the overall energy rate (over the course of a vehicle's trip through the network) relatively unchanged (i.e. less than 1 percent). In terms of future work, additional values may be tested for both acceleration and deceleration constraints for the Eco-CACC controller presented in chapter 4. In addition, the parameters of several other Eco-CACC controllers may also be tuned in order to test if overall traffic benefits are impacted by different controllers with varied acceleration and deceleration constraint values. Several additional controller designs are presented in Appendix A. An additional area of future research is evaluating the overall mobility and energy implications of various methods of platoon formation. The protocol applied in chapter 4 was an ad-hoc heterogeneous route-based platooning, involving vehicles with different routes joining the same platoon based on a triggering distance threshold being exceeded. Additional strategies include homogeneous route-based platooning, where only vehicles with the same destination join the platoon, and ordered heterogeneous

route-based platooning, where vehicles in a platoon order themselves based on the proximity of their destination. In the case of homogeneous route-based platooning, vehicles might need to actively seek out other vehicles within a certain range that have the same destination. In the case of ordered heterogeneous route-based platooning, vehicles with the furthest destination are towards the front of the platoon and vehicles with the closest destination are at the rear of the platoon. The latter strategy is designed to simplify the platoon exit maneuver by allowing the rear vehicle to simply drop off the back of the platoon. However, the simplification of the platoon exit maneuvers come at the expense of more complex platoon formation behavior. Future work may explore whether the additional disruption caused by ordering vehicles during platoon formation is offset by the simplified platoon exit maneuvers.

In chapter 5, a CV adaptive signal control optimizer with an MOE of queue length was evaluated in the context of an isolated intersection and for a corridor of 3 intersections with each intersection using the same adaptive controller in a decentralized manner. For both the isolated intersection and the signalized corridor, a volume sensitivity analysis and a demand profile sensitivity analysis were conducted. In the case of the isolated intersection, the CV queue length adaptive signal control optimizer provided benefits of -1% to 33%, and 0% to 15% for average travel time and average energy, respectively, relative to a fixed phase baseline intersection. The range of benefits relative to an intersection using Webster signal timing were 0 to 13%, and 0% to 10% for average travel time and average energy, respectively. The maximum benefits provided by using the CV queue length adaptive signal control optimizer are lower relative to the

Webster signal timing because the Webster signal timing baseline is given precise knowledge concerning the total incoming volume of hourly traffic, as well as from which direction each vehicle will approach the intersection. The fixed phase signal timing assumes that the ratios of vehicles using each traffic movement are known in advance. In spite of both of the baseline signal control strategies being given *a priori* information, the adaptive signal control strategy still provided small to moderate benefits. In addition, when compared to fixed phase signal timing, HCM signal timing, and Webster signal timing, the CV queue length adaptive signal control optimizer was the least sensitive in terms of absolute travel time.

For the case of the 3-intersection corridor, the decentralized CV queue length adaptive signal control optimization strategy provided maximum benefits of 19% and 8% in terms of average travel time and average energy, respectively, relative to coordinated fixed phase signal timing. However, the positive benefits only occurred for traffic volumes less than or equal to 4000 vph. At moderate to high traffic volumes, the performance of the CV adaptive signal control strategy was counter-productive, leading to maximum penalties of up to -22% and -8% for average travel time and average energy, respectively. A further analysis of coordinated phase vehicles progressing through the length of the corridor in both the coordinated phase signal timing baseline corridor and the CV adaptive signal control corridor revealed that the coordinated phase vehicles present in the baseline network did not benefit under decentralized adaptive signal control. In contrast, vehicles not progressing through the entire length of the corridor,

(uncoordinated phase vehicles), did benefit by using decentralized CV adaptive signal control for traffic volumes up to 4500 vph.

In terms of future work, one of the key areas for future exploration is addressing the question of which MOE or MOEs are the most practical and effective for deploying CV adaptive signal control optimizers in the field. The use of CV technology and the ability to transmit custom parameters from vehicles to intersections opens up a whole range of MOEs that were not possible under the traditional non-communicated paradigm. Although queue length was selected as the MOE of choice in this dissertation, it may be possible that a different MOE may provide even more substantial benefits. Other MOEs which may be explored include travel time, delay time, idling time, energy consumption, emissions, or even a weighted combination of multiple objectives. The advantage of using a weighted MOE, is that the behavior of a specific intersection can be customized based on the localities needs. Furthermore, the weights themselves can be dynamically adjusted based on real-time wind direction and air quality.

Another consideration reserved for future study is the effect of the size of the communication radius of vehicles approaching an intersection. Recall that the signal timing of the intersection is changed based on V2I information. If the radius is sufficiently large, then vehicles desiring to turn left may not yet have reached the left-turn bay. Consequently, the intersection would be unable to discern whether incoming vehicles were turning left, or continuing straight, until they reach the left-turn bay. One potential solution to the challenge is to have vehicles communicate their turning

intentions via wireless messages sent to the intersection. Beyond the physical technical aspects of different communication radii, there are also application features which are affected by different communication radii. One of the open areas for research is whether a CV intersection is fully able to utilize information across a large space horizon (a large communication radius). Even if a vehicle can communicate with a given intersection at a distance of 1000 meters, the intersection may not be able to make practical use of that information due to the large distance, and thus long time duration before the vehicle arrives at the intersection. As a result, the intersection may employ its own effective "receiving" radius, where it only considers vehicles which are within a certain range of the intersection.

For the signalized corridor, although the decentralized CV adaptive signal strategy did show moderate benefits, it is likely that the approach must be modified before deployment due to the current poor performance at traffic volumes greater than 4000 vph. A logical next step would be to test a centralized CV adaptive signal control strategy in which adjacent intersection communicate information which helps to optimize a stated network objective such as average network travel time. Furthermore, the strategy can be scaled to be applied to a grid network of intersections. In the case of applying a centralized CV adaptive signal control strategy to a grid network, a given intersection would communicate with up to 4 adjacent intersections.

In chapter 6, a novel eco-friendly UAID, the truncated square intersection, was derived, designed, and simulated. The truncated square intersection was compared to

existing UAIDs such as the round-about and the PFI as well as the 4-phase conventional 4-leg intersection. A series of 4 sensitivity analyses were conducted, including a volume sensitivity analysis, a major/minor ratio sensitivity analysis, a demand profile sensitivity analysis, and a left-turn demand sensitivity analysis. The volume sensitivity analysis revealed that of the intersections tested, only the PFI has a greater capacity. The PFI simulation network used had a capacity of roughly 8500 vph. In contrast, the truncated square intersection simulation network had a capacity of roughly 7500 vph. Conventional 4-leg intersections, such as the one modeled in simulation, have a capacity of between 5000 and 6000 vph. Although round-abouts tend to perform well at relatively low traffic volumes (less than 2500 vph), their capacity is less than a conventional 4-leg intersection. The capacity of the 3-lane round-about modeled in simulation was about 3500 vph. Therefore, the truncated square intersection improves upon nearly every existing intersection design in terms of intersection capacity.

The truncated square intersection simulated provided a 35% to 75%, and 24% to 41% benefit in terms of average travel time and average energy, respectively, relative to a conventional 4-leg intersection. Likewise, the benefits of the truncated square intersection over the round-about are 21% to 95%, and 28% to 53% in terms of average travel time and average energy, respectively. Finally, the maximum benefit of the truncated square intersection over the PFI is 33% for both average travel time and average energy. It is also worth pointing out that the truncated square intersection also provides emissions benefits relative to the three baseline intersections modeled.

The truncated square intersection is the first known UAID to provide an LOS of A at any traffic volume. An LOS of A corresponds to less than 10 seconds of control delay. Although the PFI provides a greater capacity, the LOS of the PFI simulated was at best level C. Not only does the truncated square intersection provide an LOS of A at a single volume, but the truncated square intersection achieved an LOS of A for traffic volumes up to 6500 vph.

In the case of the major/minor ratio, demand profile, and left-turn demand sensitivity analyses, the truncated square intersection was shown to be the least sensitive relative to the three baseline intersections simulated. Consequently, the truncated square intersection is robust to a wide variety of OD matrices, and is also robust to temporal variations in demand.

The advantages and disadvantages of building a new city based around the truncated square intersection may be categorized in the categories of operational, safety, environmental, fiscal, flexibility, innovation, and public reaction pros and cons. A number of the operational benefits of the truncated square intersection have been previously summarized including providing additional capacity, (relative to conventional 4-leg intersections and round-abouts), providing the best possible LOS of A at a large range of traffic volumes, providing benefits in reducing travel time, energy, and emissions, and providing a robust intersection that is capable of handling a wide range of potential vehicle arrival patterns, (i.e. as specified by OD matrices). Furthermore, a city built around a truncated square intersection would have no need of traffic lights, traffic

controllers, or traffic signal optimizers. Instead of Traffic Management Centers (TMCs) managing the traffic by ceaselessly measuring traffic volumes on various roads in order to adjust signal timing, TMCs could transition to becoming Traffic Monitoring Centers, in which the primary concern would be to make sure that there are not too many vehicles entering the city at any given time. One of the operational disadvantages of the truncated square intersection is that although it does have a large capacity, when the capacity is exceeded, gridlock is inevitable. As a result, the addition of meters similar to ramp meters on freeways may be located around the periphery of the city in order to regulate the total volume of traffic entering the city.

In terms of safety, the truncated square intersection eliminates the most severe crash types, such as t-bone collisions, from occurring. The most significant safety advantage of the truncated square intersection is that pedestrians are removed from the path of oncoming vehicles. The segregation of pedestrian and vehicular traffic flows improves the mobility and safety of each flow. One of the safety disadvantages of the truncated square intersection is the likelihood of increasing collisions due to vehicle weaving during lane changes. However, this disadvantage may become temporary with the prospect of automated lane changing by automated vehicles.

The environmental benefits afforded by the truncated square intersection are substantial enough to merit the classification of an Eco-UAID. A substantial portion of the energy and emissions benefits are due to vehicles spending less time on the road due to the superior mobility performance of the truncated square intersection. The remaining

balance of energy and emissions benefits is due to the reduction in the severity and frequency of acceleration and deceleration events. Drivers of conventional ICE vehicles will benefit in terms of saving fuel, and electric vehicle drivers will benefit in terms of saving energy. A city incorporating a truncated square design for traffic intersections might be included in a regional eco-zone where emissions are strictly regulated and minimized.

The fiscal merits of the truncated square intersection design were evaluated based on a detailed Benefit-to-Cost ratio Analysis. For one particular set of design parameters, the benefit-to-cost ratio was estimated at 12.5. The projected operational lifetime monetary benefit was over 157 million for a city consisting of 100 traffic intersections. Due to the absence of traffic lights, the operational and maintenance costs are substantially lower than in a conventional city setting. The reduction in travel time leads to a savings in man-hours. Another reason why the benefit-to-cost ratio is high is because of the anticipated reduction in fatalities and injuries. There are numerous and extensive costs associated with traffic collisions and especially with traffic fatalities. One of the financial hurdles is that the construction costs of a truncated intersection are anticipated to be higher than the construction costs of a conventional 4-leg intersection. An alternative method of construction would be to modify an existing city to achieve the rounded rectified square network design, which is a special case of the truncated square network design.

Another inherent advantage of the truncated square design is its flexibility in being readily applied to several transportation contexts. By adjusting the design parameter values appropriately, the truncated square design may be utilized for an isolated intersection, an urban arterial traffic network, or even a freeway interchange. One of the areas in which the design is less flexible is in terms of adding additional lanes. The addition of lanes in the round-about portion of the design increases the number of lane changes a vehicle needs to make in order to reach the center of the round-about. Although the additional lane changes are possible, especially if automated, a lower number of lane changes is preferable.

In terms of public reaction, perhaps the most significant disadvantage of the truncated square intersection is the necessity of frequent lane changes. Vehicles making a through movement or left-turn movement through the truncated square intersection each make two lane changes. However, the inclusion of automated vehicles may reduce dissatisfaction regarding the number of lane changes. One of the advantageous factors which may contribute to a positive initial public reaction and a long-term favorable perception of the intersection design is that the design is neither confusing nor complicated. One of the complaints concerning existing UAIDs is that their design is markedly complex and confuses drivers.

In terms of innovation, the truncated square intersection design is the first true continuous flow intersection design. Furthermore, when the design is tiled into a city network, vehicles are able to cross from one end of the city to another, without stopping

along the way. In addition, the novel continuous flow city is also well suited for the introduction of automated vehicles. One of the benefits for automated vehicles is that they do not need to place a high priority on pedestrian detection because pedestrians will use over or underpasses instead of crosswalks. Due to the appropriate selection of turning radii, the new city is designed such that vehicles maintain nearly the same speed throughout the network. Passengers of automated vehicles may enjoy a relatively smooth and carefree passage through the city.

The continuous flow intersections and continuous flow networks presented in the dissertation offer a glimpse of what could be the future of urban transportation. However, several areas need to be more fully explored prior to constructing a new city or modifying an existing city. A theoretical optimization of the design parameters of a truncated square intersection revealed that a rounded rectified square network should provide the optimal improvement in travel time savings across the set of possible truncated square networks. Nevertheless, the theoretical optimization needs to be confirmed in microsimulation.

A truncated square intersection design was simulated in comparison to 2 UAIDs, the round-about and the PFI, and the conventional 4-leg intersection. One avenue of future research would be to compare continuous flow intersections with each of the 15 UAIDs mentioned in the background section 2.4. Although there have been numerous studies comparing UAIDs, a single study comparing all known at-grade UAIDs has yet to be conducted. Even more importantly, the environmental implications of existing UAIDs

are not yet fully understood. Based on the limited results presented in the dissertation, the hypothesis is that the environmental benefits of true continuous flow intersections such as the truncated square intersection outweigh the environmental benefits of other UAIDs due to the fundamental operational differences.

At an even more fundamental level, the question concerning which continuous flow network design is optimal remains an avenue of future research. A total of 6 continuous flow network designs were presented in the dissertation. However, numerous other continuous flow designs remain to be enumerated and evaluated. Out of the current list of 6 continuous flow networks, a particularly interesting study would be to compare the benefits between using a rounded rectified square network and a hexagonal network relative to a conventional grid network.

Additional areas of future research include quantifying the safety performance of continuous flow intersections with the use of software such as the Surrogate Safety Analysis Method (SSAM). The BCA presented in the dissertation was conducted for a single set of parameters for a truncated square intersection design. Additional BCAs may be conducted for other parameter values as well as for other continuous flow intersection designs. In terms of architecture, designs for novel 3-way or 4-way bridges or underpasses are necessary for providing non-motorist access to facilities. In addition, the bridges or underpasses will need to provide access for pedestrians, bicyclists, and wheelchairs.

One of the lessons learned from the theoretical optimization of the design parameters of the truncated square intersection is that a smaller round-about size yields a higher percentage of benefits. However, the minimum size of the city-block size round-about is dictated by the minimum time necessary to permit human drivers to successfully make lane changes. In the case of automated vehicles, the minimum time necessary for lane changes may be less than the minimum time required for human drivers to make a lane change. The development of cooperative lane-change and weaving algorithms is necessary in order to test the hypothesis. After the algorithms for automated lane-changing for automated vehicles are simulated, the design parameters for continuous flow networks can be re-adjusted to reflect the improved efficiency of automated weaving.

Considering the range of ideas presented in the dissertation, there are two main areas of future research which involve the development of multiple systems or sub-systems. First, in the context of arterial urban traffic management a comparison of the mobility, environmental, and safety performance of several city designs can be conducted. The list of cities would include a city with a continuous flow network design, a city with connected vehicles and conventional 4-leg intersections, as well as an existing city with conventional 4-leg intersections. A thorough comparison between the benefits of continuous flow network and a conventional grid network with connected vehicles should help establish whether infrastructure re-design, or infrastructure enhancement provides the best benefits. Second, the parallel development of cooperative lane-change and merge protocols in the context of freeway traffic management and the development in the context of arterial urban traffic management of continuous flow networks

conducive to automated vehicles, introduces the possibility of creating an automated transportation system. Essentially, a future automated transportation system could be the integration of an automated highway system, enabled by Eco-CACC and additional protocols, and cities with continuous flow networks.

8. References

"Highway Capacity Manual (HCM) 2010," Transportation Research Board (TRB), 2011.

"Maryland Unconventional Intersection Design (MUID)," Applied Technology and Traffic Analysis Program. Gitbook. 2015.

"MOtor Vehicle Emission Simulator (MOVES) Version 2010b," USEPA, 2011.

"Report and Order FCC-03-324," Federal Communications Committee. Feb. 2004.

"Safety Aspects of Roundabouts". U.S. Department of Transportation Federal Highway Administration.

"Signal Timing on a Shoestring". U.S. Department of Transportation Federal Highway Administration. Washington, DC: 2005.

A. Harris. *The Mathematics of Tessellation*. St. Martin's College, 2000.

A. Ivanov and A. Tuzhilin. "The Steiner ratio Gilbert–Pollak conjecture is still open: Clarification statement." Algorithmica, 62 (1-2): 630–632. 2012.

A. Stevanovic. "Adaptive Traffic Control Systems: Domestic and Foreign State of Practice – A Synthesis of Highway Practice." NCHRP Synthesis 403, 2010.

B. Chandler, M. Myers, J. Atkinson, T. Bryer, R. Retting, J. Smithline, J. T. P. Wojtkiewicz, G. B. Thomas, S. P. Venglar, S. Sunkari, B. Malone, and P. Izadpanah. "Signalized Intersections Informational Guide, Second Edition," U.S. Department of Transportation Federal Highway Administration. No. FHWA-SA-13-027. July 2013.

B. Derksen. "Steiner4pts.jpg: A solution for the Steiner problem on four points in the Euclidean plane," March 2007. https://commons.wikimedia.org/wiki/File:Steiner_4_points.svg

B. Robinson, L. Rodegerdts, W. Scarborough, W. Kittelson, R. Troutbeck, W. Brilon, L. Bondzio, K. Courage, M. Kyte, J. Mason, A. Flannery, E. Myers, J. Bunker, and G. Jacquemart. "ROUNDABOUTS: An Informational Guide," U.S. Department of Transportation Federal Highway Administration. No. FHWA-RD-00-067. June 2000.

B. van Arem, C. J. G. van Driel, and R. Visser. "The Impact of Cooperative Adaptive Cruise Control on Traffic-Flow Characteristics," IEEE Transactions on Intelligent Transportation Systems, 7(4), 429–436, 2006.

C. Nowakowski, S. O'Connell, S. Shladover, and D. Cody. "Cooperative adaptive cruise control: Driver acceptance of following gap settings less than one second," Proceedings of the Human Factors and Ergonomics Society 54th Annual Meeting, 2033-2037, 2010.

C. Pozrikidis. "An Introduction to Grids, Graphs, and Networks," Oxford University Press, NY, 2014.

C-Y. Liang and H. Peng. "String Stability Analysis of Adaptive Cruise Controlled Vehicles," JSME International Journal Series C, 43(3), 671-677, 2000.

D. Bevly, X. Cao, M. Gordon, G. Ozbilgin, D. Kari, B. Nelson, J. Woodruff, M. Barth, C. Murray, A. Kurt, K. Redmill, and U. Ozguner. "Lane Change and Merge Maneuvers for Connected and Automated Vehicles: A Survey," Submitted to IEEE Transactions on Intelligent Transportation Systems, 2015.

D. Darling. "The Universal Book of Mathematics: From Abracadabra to Zeno's Paradoxes," p. 95, 2004.

D. Kari, G. Wu, and M. Barth. "Eco-Friendly Freight Signal Priority Using Connected Vehicle Technology: A Multi-Agent Systems Approach," IEEE on Intelligent Vehicle Symposium, June 2014.

D. Levinson and A. El-Geneidy. "The minimum circuity frontier and the journey to work," Regional Science and Urban Economics, 39:6, 732-738, Nov. 2009.

D. Schrank, B. Eisele, and T. Lomax. "TTI's 2012 Urban Mobility Report," Dec. 2012.

D. Swaroop and K. Hedrick. "String Stability of Interconnected Systems," IEEE Transactions on Automatic Control, Vol. 41, No. 3, 349 – 357, 1996.

F. Browand, J. McArthur, and C. Radovich. "Fuel Saving Achieved in the Field Test of Two Tandem Trucks," PATH Research Report UCB-ITS-PRR-2004-20, 2004.

F. Bu, H-S. Tan, and J. Huang. "Design and Field Testing of A Cooperative Adaptive Cruise Control System," Proceedings of the American Control Conference, Baltimore, MD, 4616–4621, 2010.

F. D. Mier, and B. H. Romo. "Continuous Flow Intersection," US Patent No.: 5,049,000, Sept. 1991.

F. K. Hwang, D. S. Richards, and P. Winter. "The Steiner Tree Problem," Annals of Discrete Mathematics, 53, North-Holland, 1992.

G. Chlewicki. "New Interchange and Intersection Designs: The Synchronized Split-Phasing Intersection and the Diverging Diamond Interchange," Prepared for 2nd Urban Street Symposium, Anaheim, CA, 2003.

G. F. Parsons. "[Parallel Flow Intersection Diagrams,]" Quadrant Engineering, LLC.

G. F. Parsons. "Parallel Flow Vehicle Turn System for Traffic Intersections," US Patent No.: 7,135,989, Nov. 2006.

G. F. Parsons. "The parallel flow intersection: A new high capacity urban intersection," Advanced Forum on Transportation of China (AFTC 2009), 5th, Beijing, 143-150, 2009.

G. Naus *et al.* "String-Stable CACC Design and Experimental Validation: A Frequency-Domain Approach," IEEE Transactions on Vehicular Technology, 59(9), 4268–4279, 2010.

H. H. Hsu and A. Liu. "Kinematic Design for Platoon-Lane-Change Maneuvers," IEEE Transactions on Intelligent Transportation Systems, 9(1), 185-190, 2008.

H. Jula, E. B. Kosmatopoulos, and P. A. Ioannou. "Collision Avoidance Analysis for Lane Changing and Merging," IEEE Transactions on Vehicular Technology, 49(6), 2295-2308, 2000.

H. Xia, G. Wu, K. Boriboonsomsin, and M. J. Barth, "Development and evaluation of an enhanced eco-approach traffic signal application for Connected Vehicles," 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), The Hague, 296-301, 2013.

J. E. Hummer. "Unconventional left-turn alternative for urban and suburban arterials – part one," ITE Journal, 68 (9), 26–29, 1998.

J. Guo and N. Balon. "Vehicular Ad Hoc Networks and Dedicated Short Range Communication," University of Michigan, Dearborn. June 2006.

J. Hüper, G. Dervisoglu, A. Muralidharan, G. Gomez, R. Horowitz, and P. Varaiya. "Modeling and Simulation of Freeway Traffic Flow," *12th IFAC Symposium on Control in Transportation Systems*, 2009.

J. L. Ganley. "Steiner ratio," in Dictionary of Algorithms and Data Structures [online], Vreda Pieterse and Paul E. Black, eds. 17 Dec. 2004. http://www.nist.gov/dads/HTML/steinerratio.html

J. Sangster, H. Rakha, and A. Al-Kaisy. "Comparative Analysis of the Through-about, Roundabout, and Conventional Signalized Intersection Designs," Submitted to Transportation Research Board 2015 Annual Meeting. 2015.

K. Boriboonsomsin and M. Barth. "Impacts of Freeway High-Occupancy Vehicle Lane Configuration on Vehicle Emissions," Transportation Research Part D, 13(2), 112-125, 2008

L. Rodegerdts, B. Nevers, B. Robinson, J. Ringert, P. Koonce, J. Bansen, T. Nguyen, J. McGill, D. Stewart, J. Suggett, T. Neuman, N. Antonucci, K. Hardy, and K. Courage. "Signalized Intersections: Informational Guide," U.S. Department of Transportation Federal Highway Administration. No. FHWA-HRT-04-091. Aug. 2004.

M. E. Esawey and T. Sayed. "Operational performance analysis of the unconventional median U-turn design using micro-simulation." Canadian Journal of Civil Engineering, 38 (11), 1249–1261, 2011.

M. E. Esawey and T. Sayed. "Analysis of unconventional arterial intersection designs (UAIDs): state-of-the-art methodologies and future research directions, Transportmetrica A: Transport Science," 9:10, 860-895, 2013.

M. Niazi and A. Hussain. "Agent-based Computing from Multi-agent Systems to Agent-Based Models: A Visual Survey," Springer Scientometrics, 89(2), 479-499, 2011.

NEMA: National Electrical Manufacturers Association, The Association of Electrical Equipment and Medical Imaging Manufacturers. http://www.nema.org/pages/default.aspx

P. Koonce, L. Rodegerdts, K. Lee, S. Quayle, S. Beaird, C. Braud, J. Bonneson, P. Tarnoff, and T. Urbanik. "Traffic Signal Timing Manual," U.S. Department of Transportation Federal Highway Administration. No. FHWA-HOP-08-024. June 2008.

Parks, *et al*.. "Semiregular Tilings." in *A Mathematical View of Our World*, 1st ed. Belmont: Thomson, 2007.

Q. Xu and R. Sengupta. "Simulation, Analysis, and Comparison of ACC and CACC in Highway Merging Control," Intelligent Vehicles Symposium, 237-242, 2003.

Quadstone, Paramics, 2016. http://www.paramics-online.com/.

R. C. Larson and A. R. Odoni. "Urban Operations Research," Prentice-Hall, NJ, 1981.

R. Gordon. "Methodologies to Measure and Quantify Transportation Management Center Benefits: Final Synthesis Report," U.S. Department of Transportation Federal Highway Administration. No. FHWA-HRT-12-054. Dec. 2012.

R. J. Caudill, and W. L. Garrard. "Vehicle-follower Longitudinal Control for Automated Transit Vehicles," Journal of Dynamics Systems, Measurement, and Control, 241-248, 1977.

R. M. Ziff and C. R. Scullard. "Martini.png," Nov. 2006. https://commons.wikimedia.org/wiki/File:Martini.png

SAE J2735 “Dedicated Short Range Communications (DSRC) Message Set Dictionary,” Nov. 2009.

S. E. Shladover, D. Su, and X. Lu. “Impacts of Cooperative Adaptive Cruise Control on Freeway Traffic Flow.” Transportation Research Record, No. 2324, 63-70, 2012.

S. Öncü, J. Ploeg, N. van de Wouw, and H. Nijmeijer. “Cooperative Adaptive Cruise Control: Network-Aware Analysis of String Stability,” IEEE Transactions on Intelligent Transportation Systems 15 (4), 1527-1537, 2014.

S. Tsugawa, S. Kato, and K. Aoki. “An Automated Truck Platoon for Energy Saving,” IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.

T. C. Hales. “The Honeycomb Conjecture,” Discrete and Computational Geometry, 25 (1): 1-22. 2001.

Transportation Research Board. “HCM 2000,” National Academy of Sciences. 2000.

V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura. “Cooperative Adaptive Cruise Control in Real Traffic Situations,” IEEE Transactions on Intelligent Transportation Systems 15 (1), 296-305, 2014.

W. Burghout, H. Koutsopoulos, and I. Andréasson. “Hybrid mesoscopic-microscopic traffic simulation,” The World Conference on Transportation Research, 2004.

W. Chee and M. Tomizuka. “Vehicle Lane Change Maneuver in Automated Highway Systems,” Final Report for California Partners for Advanced Transit and Highways (PATH) program, 1994.

W. Hughes, R. Jagannathan, D. Sengupta, and J. Hummer. “Alternative Intersections/Interchanges: Informational Report (AIIR),” U.S. Department of Transportation Federal Highway Administration. No. FHWA-HRT-09-060. April 2010.

X-Y. Lu, H-S. Tan, S. E. Shladover, and K. Hedrick. “Automated Vehicle Merging Maneuver Implementation for AHS,” Vehicle System Dynamics, 41(2), 85-107, 2004.

X-Y. Lu, K. Hedrick, and M. Drew. “ACC/CACC-Control Design, Stability and Robust Performance,” Proceedings of the American Control Conference, Anchorage, AK, 2000.

Appendix A: Candidates for Eco-CACC Controllers

There are at least three defining features of any CACC controller: 1) the regulator type (e.g. spatial or temporal), 2) the controller structure (e.g. P, PD, or PID), and 3) the constraints incorporated. In the following table, the controller structure is assumed to be based on proportional control. Future work will look at comparing various control structures.

Table A.1 Eco-CACC Controller Options

Constraints \ Regulator Type	Spatial	Temporal
No Constraints	Figure A.1	Figure A.3
+ Acceleration Constraints	Figure A.2	Figure A.4
+ Spatial Constraints		

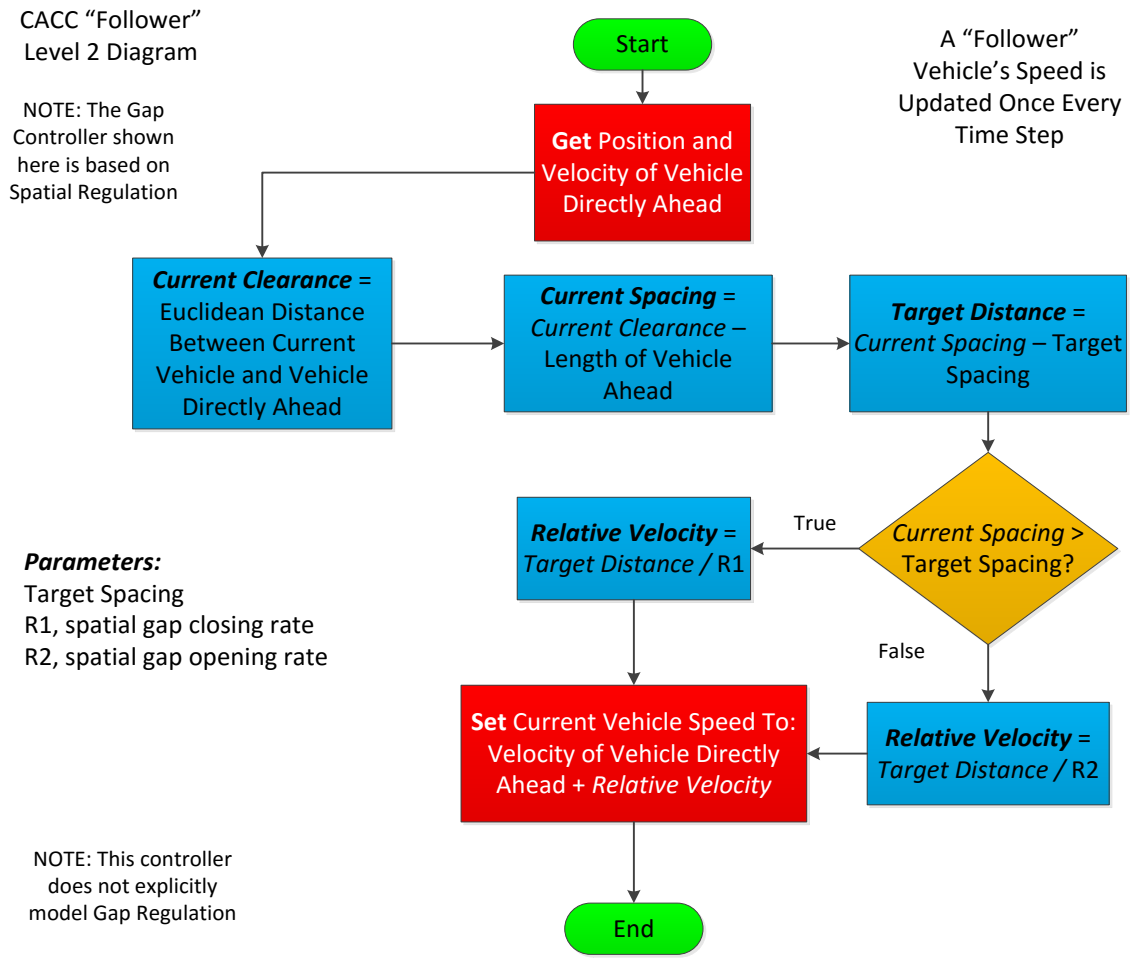

Figure A.1: Spatial Regulator

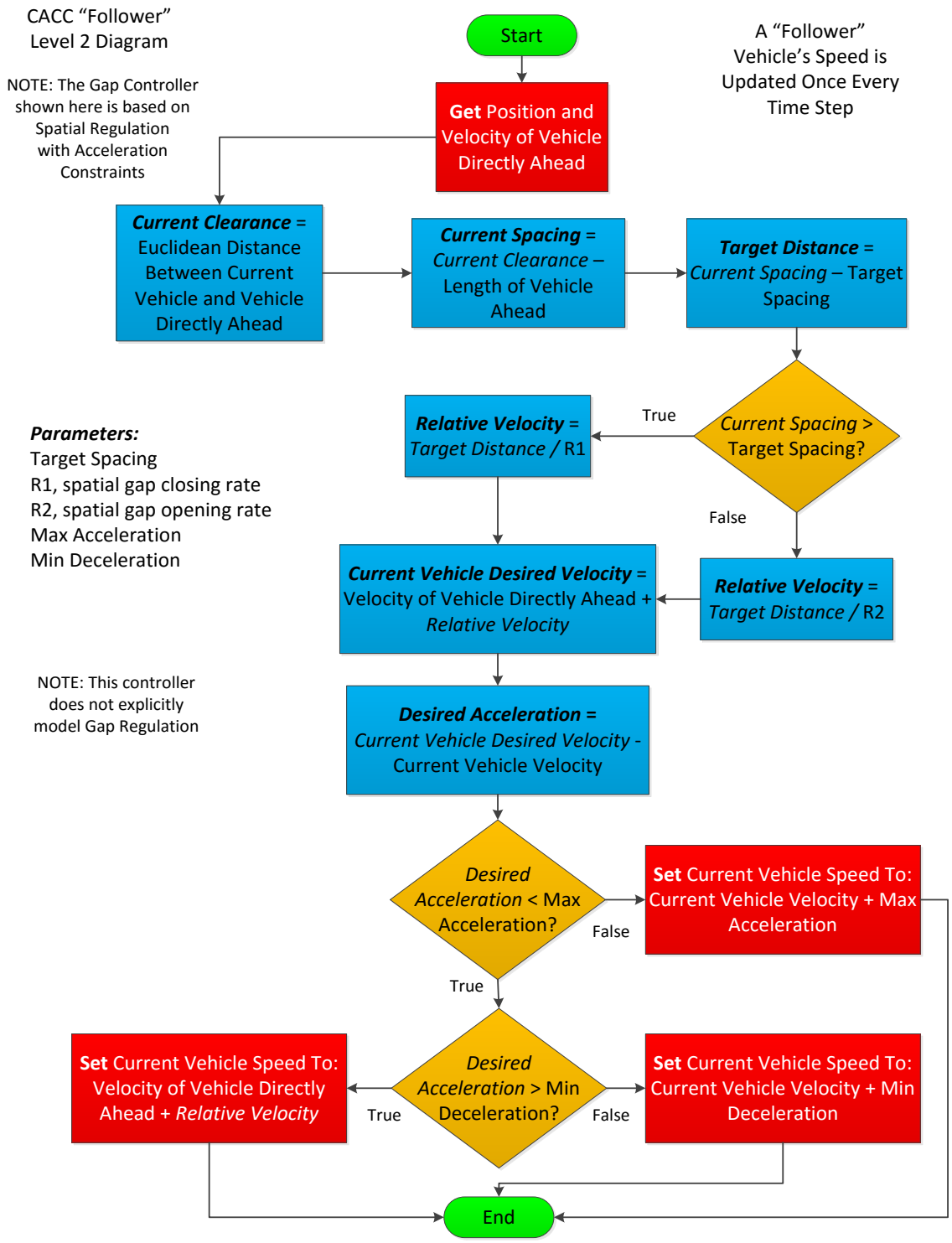

Figure A.2: Spatial Regulator with Acceleration Constraints

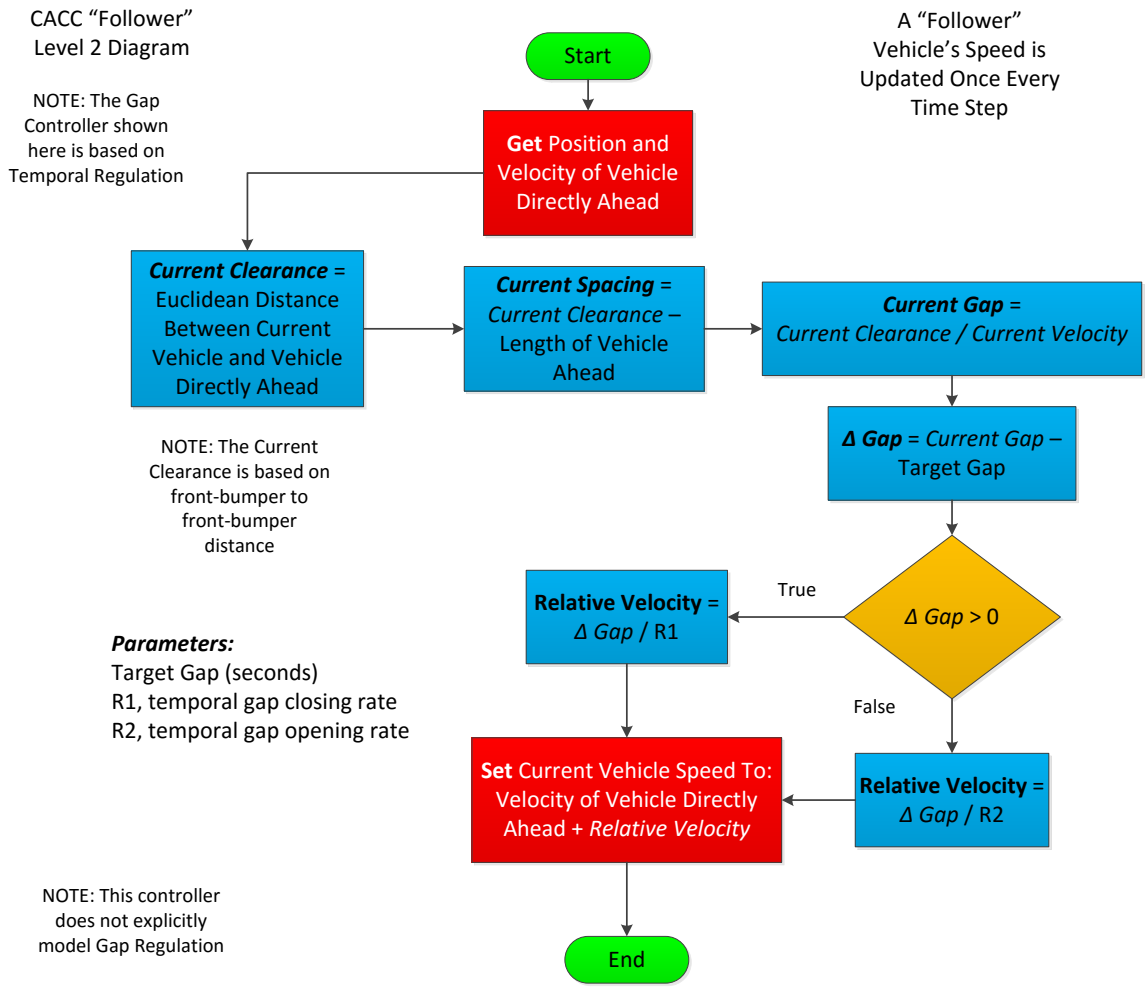

Figure A.3: Temporal Regulator

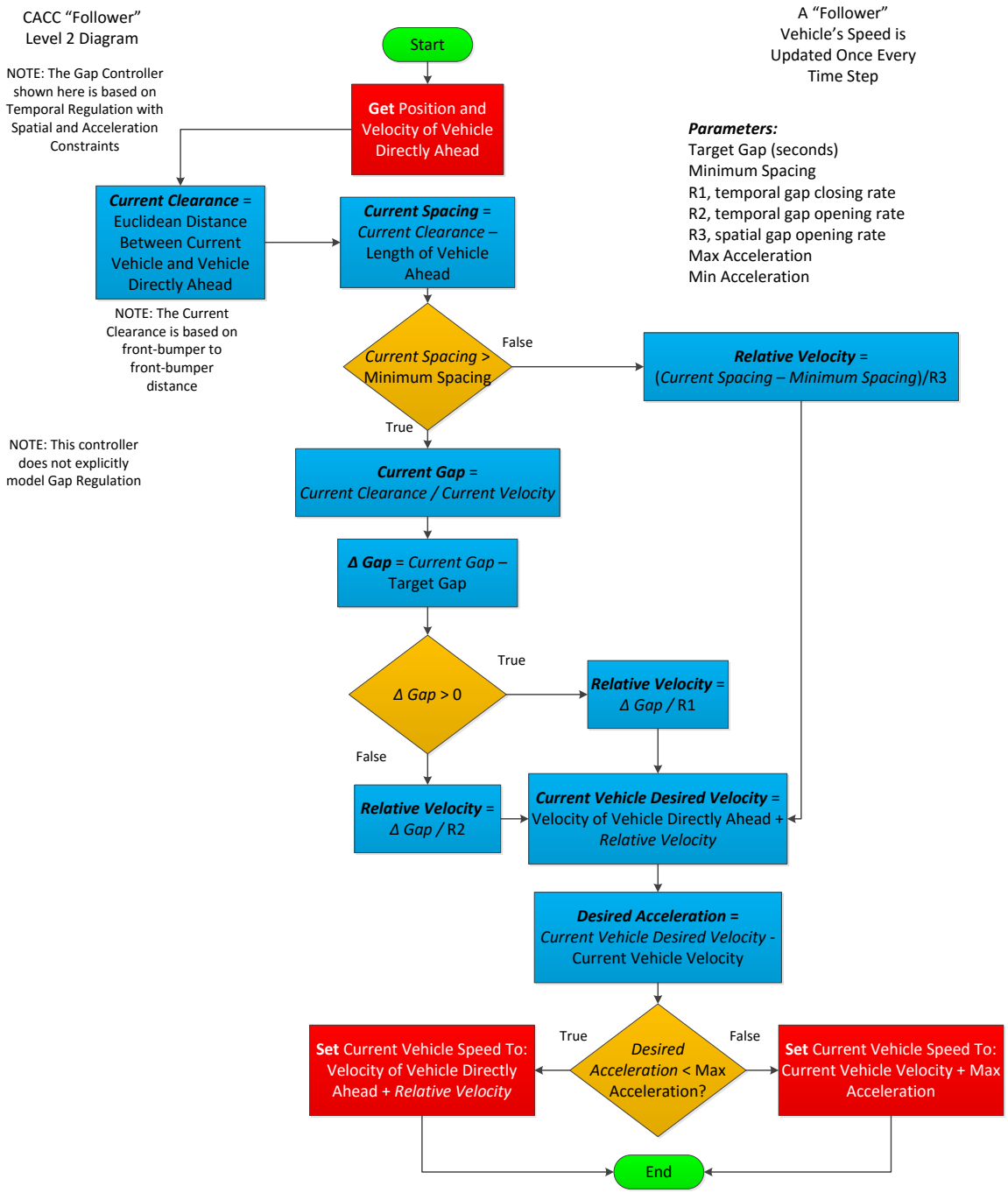

Figure A.4: Temporal Regulator with Spatial and Acceleration Constraints

Appendix B: Freeway CACC Code

The following code is C/Paramics plugin code for Freeway CACC.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "programmer.h"
#include <windows.h>
#include <assert.h>

#define MIN(a,b)        ((a) < (b) ? (a) : (b))
#define MAX(a,b)        ((a) > (b) ? (a) : (b))
#define NDEBUG //comment out this line to activate assert statements

typedef struct veh_profile VEHICLE_DATA;
struct veh_profile
{
	float A; // rolling coefficient in kW*sec/meter
	float B; // rotation coefficient in kW*sec^2/meter^2
	float C; // drag coefficient in kW*sec^3/meter^3
	float M; // vehicle source mass in metric tons
	float f; // fixed mass factor in metric tons
	int sourceType; // vehicle source type
	int ID; // vehicle ID
	float theta; // road grade angle
	float vel[4]; // 3-second velocities
	float acc[4]; // 3-second accelerations
	int linkID[4]; // 3-second link IDs
	float VSP[4]; // 3-second VSPs
	int mode[4]; // 3-second modes
	float tm[4]; // 3-second time stamps
	int first2sec; // the first 2 seconds vehicle enters the network
	float energy[4], CO2[4], CO[4], HC[4], NOx[4], PM[4]; // 3-second emission
data
	int origin; // vehicle's origin zone
	int dest; // vehicle's destination zone
	float x_destNode; // x coordinate of start node of vehicle's destination
zone
	float y_destNode; // y coordinate of start node of vehicle's destination
zone
	float tripDist; // vehicle's total trip distance
	int bound; // 1: northbound; 2: southbound: 0 otherwise
	int CACCstatus; // 1: leader; 0: uninitialized; -1: follower
	int departLaneIndex; // test variable for restricting all lane changes
(with the exception of changing lanes to make it through a bottle neck)
	int platoonLaneIndex; // test variable, appropriate lane index for
platooning (changes based on current and next link geometry)
	int necessaryLaneChange; // 1 = lane change necessary due to roadway
geometry (i.e. bottle neck),		...currently unused
```

```
	int laneDropApproaching; // 1 if next link involves a lane drop, 0
otherwise
	VEHICLE* vehicleAhead; //pointer to vehicle directly ahead
	LINK* currentLink; //pointer to most recent current link
	float prevSpeed; //used to calculate acceleration
	float vehTargetSpacing; // used for all CACC manuevers
	int merging; // 0 if not merging, or conditions for merging have not been
met, 1 = Stage 1 of merging: longitudinal relative positioning, 2 = Stage 2 of
merging: lateral (lane-change) manuever
	int splitting; // 0 if not splitting, 1 if the vehicle is the lead vehicle
of the rear platoon fragment
	VEHICLE* vehicleAdjacentBehindP; // pointer to vehicle on adjacent lane
behind the current vehicle
	VEHICLE* vehicleAdjacentAheadP; // pointer to vehicle on adjacent lane
ahead of the current vehicle
	VEHICLE* vehicleBehindP; // pointer to vehicle directly behind the current
vehicle, ...used for splitting manuevers
	int EQUIPPED_FLAG; //1 = equipped, 0 = unequipped
	int PARTICIPATION_FLAG; //1 = participating, 0 = not participating (any
longer)  ...this flag is used to permit a vehicle to leave a platoon, or to
prevent platoon formation in the first place
	int PRIOR_PARTICIPATION_FLAG; //used to detect a change in participation
(from prior time step to the current time step)
	int MAINSTREAM_FLAG; //1 = vehicle is now merged with mainstream freeway
traffic, 0 = vehicle has not yet merged with mainstream freeway traffic ...this
flag is used to set participation flag for vehicles entering freeway from an on-
ramp
	int EXIT_APPROACHING_FLAG; // 1 if exit is approaching, 0 otherwise
	int PARTICIPATED_FLAG; //1 if vehicle participated in a platoon at any
point in time during their trip, 0 otherwise
	float total_energy; //trip energy for ego-vehicle
	float total_dist; //trip distance for ego-vehicle (may replace tripDist)
};

//Parameters
double PenetrationRate = 1.0;	// penetration rate of technology, (where
percentage, max 100%, is expressed in decimal form)
const float followThreshold = 40; // threshold of following distance
const float targetSpacing = 5; // meters

const int dedicatedLanes = 1; // number of dedicated lanes (starting from left
moving to right)

//Constants
const float GRAV = 9.8; // gravity coefficient in meter/second^2
const float INDEX = 2.23693629; // conversion index from m/s to mph
const float meter2mile = 0.000621371;
const float vehicleLength = 3.9990; //4.8768 was for SUMO; // can replact this by
using the qpg_VHC_length(VEHICLE* vehicle) function (needed for multiple vehicle
types)
float DELTA = 0.1; // default small value, later modified in qpx_NET_postOpen()
// for MOVES
float moves[63][41][6]; // moves[regClass][opMode][emissionCategory]
int modeBins[50] = {0}; // opMode distribution
```

```
// overall output data
char out1Path[150]; // file "data_sbs.dat"
char out2Path[150]; // file "MOVES_sum.dat"
char out3Path[150]; // file "opMode.dat"
char out4Path[150]; // file "VSP.dat"
char out5Path[150]; // file "TT.dat"
// for equipped vehicle only
char out11Path[150]; // file "data_sbs_e.dat"
char out21Path[150]; // file "MOVES_sum_e.dat"
char out31Path[150]; // file "opMode_e.dat"
char out41Path[150]; // file "VSP_e.dat"
char out51Path[150]; // file "TT_e.dat"
// for unequipped vehicle only
char out12Path[150]; // file "data_sbs_n.dat"
char out22Path[150]; // file "MOVES_sum_n.dat"
char out32Path[150]; // file "opMode_n.dat"
char out42Path[150]; // file "VSP_n.dat"
char out52Path[150]; // file "TT_n.dat"


//int timeStamp_sbs[10000000]; // second by second vehcle ID
//int vehID_sbs[10000000]; // second by second vehcle ID
//int linkID_sbs[10000000]; // second by second link ID
//float VSP_sbs[10000000]; // second by second VSP
//float speed_sbs[10000000]; // second by second speed
//float acc_sbs[10000000]; // second by second acc
//int vehType_sbs[10000000]; // vehicle type mapping to speed_sbs
//int origin_sbs[10000000];
//int dest_sbs[10000000];
//int counter_sbs; // counter for VSP_sbs
float TT_veh[500000]; // travel time of each vehicle

int counter_released = 0;
int counter_released_e = 0;
int counter_released_n = 0;

int counter_arrived = 0;
int counter_arrived_e = 0;
int counter_arrived_n = 0;

int counter_participated = 0; //number of equipped vehicles which enter the
dedicated lane(s)
int counter_arrived_d_mainstream = 0;
int counter_arrived_nd_mainstream = 0;

// aggregated data
// define variables for overall traffic
double VMT = 0, VHT = 0, dist_sum = 0, energy_sum = 0, CO2_sum = 0, CO_sum = 0,
HC_sum = 0, NOx_sum = 0, PM_sum = 0;
// define variables for equipped vehicles only
double VMT_e = 0, VHT_e = 0, dist_sum_e = 0, energy_sum_e = 0, CO2_sum_e = 0,
CO_sum_e = 0, HC_sum_e = 0, NOx_sum_e = 0, PM_sum_e = 0;
// define variables for non-equipped vehicles only
```

```
double VMT_n = 0, VHT_n = 0, dist_sum_n = 0, energy_sum_n = 0, CO2_sum_n = 0,
CO_sum_n = 0, HC_sum_n = 0, NOx_sum_n = 0, PM_sum_n = 0;

// define variables for dedicated lane only
double VMT_d = 0, VHT_d = 0, dist_sum_d = 0, energy_sum_d = 0, CO2_sum_d = 0,
CO_sum_d = 0, HC_sum_d = 0, NOx_sum_d = 0, PM_sum_d = 0;
// define variable for non-dedicated lanes only
double VMT_nd = 0, VHT_nd = 0, dist_sum_nd = 0, energy_sum_nd = 0, CO2_sum_nd = 0,
CO_sum_nd = 0, HC_sum_nd = 0, NOx_sum_nd = 0, PM_sum_nd = 0;

double VHT_d_mainstream = 0, VHT_nd_mainstream = 0;
//used for dedicated lane vehicle statistics
double dist_sum_d_mainstream = 0;
double energy_sum_d_mainstream = 0;
// used for non-dedicated lane vehicle statistics
double dist_sum_nd_mainstream = 0;
double energy_sum_nd_mainstream = 0;

//Acceleration/Deceleration Constraints
float max_accel_mpss = 3.5; //3.5 m/s^2 (NOTE: in this case, the selected values
were set to match the selected profile (overall max), other values may be set
(based on speed, or platoon member comfort))
float min_decel_mpss = -7.5; //-7.5 m/s^2

//DEBUG global variables
int DEBUG_FLAG = 0;

void qpx_NET_postOpen()
{
	int i, j, k, md;
	int reg[13] = {11,21,31,32,41,42,43,51,52,53,54,61,62}; // source types
	char *path, *outPath;
	char inPath[150] = "";
	FILE *fin;

	DELTA = 0.5*qpg_CFG_timeStep();
	path = qpg_NET_dataPath();
	outPath = qpg_NET_statsPath();
	strcpy(inPath, path);
	strcpy(out1Path, outPath);
	strcpy(out2Path, outPath);
	strcpy(out3Path, outPath);
	strcpy(out4Path, outPath);
	strcpy(out5Path, outPath);
	strcpy(out11Path, outPath);
	strcpy(out21Path, outPath);
	strcpy(out31Path, outPath);
	strcpy(out41Path, outPath);
	strcpy(out51Path, outPath);
	strcpy(out12Path, outPath);
	strcpy(out22Path, outPath);
	strcpy(out32Path, outPath);
	strcpy(out42Path, outPath);
	strcpy(out52Path, outPath);
```

```
	strcat(inPath, "/sourceTypes_2005.txt");
	strcat(out1Path, "/data_sbs.dat");
	strcat(out2Path, "/MOVES_sum.dat");
	strcat(out3Path, "/opMode.dat");
	strcat(out4Path, "/VSP.dat");
	strcat(out5Path, "/TT.dat");
	strcat(out11Path, "/data_sbs_e.dat");
	strcat(out21Path, "/MOVES_sum_e.dat");
	strcat(out31Path, "/opMode_e.dat");
	strcat(out41Path, "/VSP_e.dat");
	strcat(out51Path, "/TT_e.dat");
	strcat(out12Path, "/data_sbs_n.dat");
	strcat(out22Path, "/MOVES_sum_n.dat");
	strcat(out32Path, "/opMode_n.dat");
	strcat(out42Path, "/VSP_n.dat");
	strcat(out52Path, "/TT_n.dat");

	fin = fopen(inPath, "r");
	if (fin == NULL)
		qps_GUI_printf("Couldn't open file. (simulation started)");
	fscanf(fin, "%d,", &md); // md: opMode
	for (i = 0; i < 13; i++) // i: regClass
		for (j = 0; j < 41; j++) // j: opMode
			if (md == j)
			{
				fscanf(fin, "%f,%f,%f,%f,%f,%f", &moves[reg[i]][j][0],
&moves[reg[i]][j][1], &moves[reg[i]][j][2], &moves[reg[i]][j][3],
&moves[reg[i]][j][4], &moves[reg[i]][j][5]);
				fscanf(fin, "%d,", &md);
			}
	for (j = 0; j < 41; j++) {
		for (k = 0; k <= 5; k++) {
			moves[20][j][k] = moves[21][j][k];
		}
	}
	fclose(fin);
}

int opMode(float vsp, float speed, float acc)
{
	speed = speed*INDEX;
	acc = acc*INDEX;
	if (acc <= -2) return 0;

	if (speed < 1 && speed >= -1) return 1;
	if (speed < 25 && speed >= 0) //this second value should be 1, (error in
Haitao's code, never reached though, due to previous statement)
	{
		if (vsp < 0) return 11;
		if (vsp < 3) return 12;
		if (vsp < 6) return 13;
		if (vsp < 9) return 14;
		if (vsp < 12) return 15;
```

```
                return 16;}
        else if (speed < 50) {
                if (vsp < 0) return 21;
                if (vsp < 3) return 22;
                if (vsp < 6) return 23;
                if (vsp < 9) return 24;
                if (vsp < 12) return 25;
                if (vsp < 18) return 27;
                if (vsp < 24) return 28;
                if (vsp < 30) return 29;
                return 30;}
        else {
                if (vsp < 6) return 33;
                if (vsp < 12) return 35;
                if (vsp < 18) return 37;
                if (vsp < 24) return 38;
                if (vsp < 30) return 39;
                return 40;}
}

void updateVehAttributes(VEHICLE_DATA* myVeh)
{
        if(myVeh->sourceType == 11)
        {
                myVeh->A = 0.0251;
         myVeh->B = 0;
         myVeh->C = 0.000315;
         myVeh->M = 0.285;
                myVeh->f = 0.285;
                //myVeh->regClass = 10;
        }
        else if(myVeh->sourceType == 21 || myVeh->sourceType == 20)
        {
                myVeh->A = 0.156461;
         myVeh->B = 0.002002;
         myVeh->C = 0.000493;
         myVeh->M = 1.4788;
                myVeh->f = 1.4788;
                //myVeh->regClass = 20;
        }
        else if(myVeh->sourceType == 31)
        {
                myVeh->A = 0.22112;
         myVeh->B = 0.002838;
         myVeh->C = 0.000698;
         myVeh->M = 1.86686;
                myVeh->f = 1.86686;
                //myVeh->regClass = 30;
        }
        else if(myVeh->sourceType == 32)
        {
                myVeh->A = 0.235008;
         myVeh->B = 0.003039;
         myVeh->C = 0.000748;
```

```
 myVeh->M = 2.05979;
       myVeh->f = 2.05979;
       //myVeh->regClass = 30;
}
else if(myVeh->sourceType == 41)
{
       myVeh->A = 1.29515;
 myVeh->B = 0;
 myVeh->C = 0.003715;
 myVeh->M = 19.5937;
       myVeh->f = 17.1;
       //myVeh->regClass = 48;
}
else if(myVeh->sourceType == 42)
{
       myVeh->A = 1.0944;
 myVeh->B = 0;
 myVeh->C = 0.003587;
 myVeh->M = 16.556;
       myVeh->f = 17.1;
       //myVeh->regClass = 48;
}
else if(myVeh->sourceType == 43)
{
       myVeh->A = 0.746718;
 myVeh->B = 0;
 myVeh->C = 0.002176;
 myVeh->M = 9.06989;
       myVeh->f = 17.1;
       //myVeh->regClass = 46;
}
else if(myVeh->sourceType == 51)
{
       myVeh->A = 1.41705;
 myVeh->B = 0;
 myVeh->C = 0.003572;
 myVeh->M = 20.6845;
       myVeh->f = 17.1;
       //myVeh->regClass = 47;
}
else if(myVeh->sourceType == 52)
{
       myVeh->A = 0.561933;
 myVeh->B = 0;
 myVeh->C = 0.001603;
 myVeh->M = 7.64159;
       myVeh->f = 17.1;
       //myVeh->regClass = 42;
}
else if(myVeh->sourceType == 53)
{
       myVeh->A = 0.498699;
 myVeh->B = 0;
 myVeh->C = 0.001474;
```

```
 myVeh->M = 6.25047;
        myVeh->f = 17.1;
        //myVeh->regClass = 41;
    }
    else if(myVeh->sourceType == 54)
    {
        myVeh->A = 0.617371;
 myVeh->B = 0;
 myVeh->C = 0.002105;
 myVeh->M = 6.73483;
        myVeh->f = 17.1;
        //myVeh->regClass = 42;
    }
    else if(myVeh->sourceType == 61)
    {
        myVeh->A = 1.96354;
 myVeh->B = 0;
 myVeh->C = 0.004031;
 myVeh->M = 29.3275;
        myVeh->f = 17.1;
        //myVeh->regClass = 47;
    }
    else //if(myVeh->sourceType == 62)
    {
        myVeh->A = 2.08126;
 myVeh->B = 0;
 myVeh->C = 0.004188;
 myVeh->M = 31.4038;
        myVeh->f = 17.1;
        //myVeh->regClass = 47;
    }
}

void  qpx_VHC_release(VEHICLE* Vp)
{
    VEHICLE_DATA *myVeh = calloc(1, sizeof(VEHICLE_DATA));
    double rn;
    float x5 = 0; // used for destination node location
    float y5 = 0; // used for destination node location
    float z5 = 0; //unused
    counter_released++;

    rn = (double)rand()/(double)RAND_MAX; // this may not achieve the exact
desired penetration rate, and will vary with seed number (DK)

    if (rn <= PenetrationRate)
    {
        myVeh->EQUIPPED_FLAG = 1;
        //Method 1: All-lanes version
        //myVeh->origin = qpg_VHC_origin(Vp);
        //if (myVeh->origin == 4 || myVeh->origin == 6) //can generalize
this later...this specifies that vehicles on on-ramps are not permitted to
participate in platoon formation until after merging with traffic (can add more
sophisticated platoon merging in later project)
```

```
		//{
		//	myVeh->PARTICIPATION_FLAG = 0; //this is set to 1 after
merging with mainline traffic
		//	myVeh->PRIOR_PARTICIPATION_FLAG = 0;
		//	myVeh->MAINSTREAM_FLAG = 0;
		//	//qps_GUI_printf("Vehicle %d trip %d -> %d on link %s;
PARTICIPATION_FLAG = 0\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)));
		//}
		//else
		//{
		//	myVeh->PARTICIPATION_FLAG = 1; //can change this later to
model equipped vehicles not participating in platoons
		//	myVeh->PRIOR_PARTICIPATION_FLAG = 1;
		//	myVeh->MAINSTREAM_FLAG = 1;
		//}

		//Method 2: Dedicated lane version, has vehicles starting out as not
participating, lane is later checked to set participation flag to 1
		myVeh->PARTICIPATION_FLAG = 0; //this is set to 1 if vehicle is in a
dedicated lane, and reset to zero if it leaves the dedicated lane
		myVeh->PRIOR_PARTICIPATION_FLAG = 0;

		counter_released_e += 1;
	}
	else
	{
		myVeh->EQUIPPED_FLAG = 0;
		myVeh->PARTICIPATION_FLAG = 0;
		myVeh->PRIOR_PARTICIPATION_FLAG = 0;
		counter_released_n += 1;
	}
	myVeh->EXIT_APPROACHING_FLAG = 0; //initially zero, set to one based on
selected condition of proximity to exit

	myVeh->PARTICIPATED_FLAG = 0; //initially zero, permanently switched to one
if PARTICIPATION_FLAG is set to one

	// initialize data for MOVES model and CACC
	myVeh->ID = qpg_VHC_uniqueID(Vp);
	myVeh->sourceType = qpg_VHC_type(Vp); // emission tables directly related
to source types
	myVeh->theta = 0;
	myVeh->first2sec = 1;
	myVeh->origin = qpg_VHC_origin(Vp);
	myVeh->dest = qpg_VHC_destination(Vp);
	myVeh->tripDist = 0;//qpg_RTR_distanceRemaining(qpg_VHC_link(Vp), Vp);
//this function is bugged, and is not working with the on/off ramp network
	myVeh->CACCstatus = 0; // default state (not in a CACC platoon)
	myVeh->departLaneIndex = qpg_VHC_lane(Vp); // test
	myVeh->platoonLaneIndex = qpg_VHC_lane(Vp); // test, by default, at
initialization, platoonLaneIndex = departLaneIndex
	myVeh->necessaryLaneChange = 0; // test
	myVeh->vehicleAhead = NULL; // test
```

```
	myVeh->currentLink = qpg_VHC_link(Vp); // test
	myVeh->laneDropApproaching = 0; // test, default initialization is false
(no impending lane drop)
	myVeh->prevSpeed = 0; // used to calculate acceleration
	myVeh->vehTargetSpacing = targetSpacing; // default target spacing is
targetSpacing parameter
	myVeh->merging = 0; //default is 0 (not merging)
	myVeh->splitting = 0; //default is 0 (not splitting)
	myVeh->vehicleAdjacentBehindP = NULL; //default
	myVeh->vehicleAdjacentAheadP = NULL; //default
	myVeh->vehicleBehindP = NULL; //default

	//retrieve destination node position
	qpg_POS_node(qpg_LNK_nodeStart(qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1)),
&x5, &y5, &z5);
	myVeh->x_destNode = x5;
	myVeh->y_destNode = y5;

	myVeh->total_energy = 0; //default initialization, aggregated when vehicle
is in the network

	updateVehAttributes(myVeh);
	qps_VHC_userdata(Vp, (VEHICLE_DATA *)myVeh);

	//color vehicle based on destination zone
	if (myVeh->dest == 3) //first off-ramp
	{
		qps_DRW_vehicleColour(Vp, 4278190335); //red 4278190335 (0xFF0000FF)
//medium green 4280791591 (0xFF27B227) //forest green 4279857945 (0xFF197319)
//light green 4278255360 (OxFF00FF00)
	}
	else if (myVeh->dest == 5) // second off-ramp
	{
		qps_DRW_vehicleColour(Vp, 4294902015); //magenta 4294902015
(0xFFFF00FF) //cyan 4294967040 (0xFFFFFF00) //yellow 4278255615 (0xFF00FFFF)
//blue 4294901760 (OxFFFF0000)
	}
	else if (myVeh->dest == 2) // end of network
	{
		qps_DRW_vehicleColour(Vp, 4278255360); //light green
	}
}

void calEmissions(VEHICLE_DATA* myVeh, float currTime, float vel_c, int linkid)
{
	int i;
	// shift 4-second data
	for (i = 0; i < 3; i++)
	{
		myVeh->vel[i] = myVeh->vel[i+1];
		myVeh->acc[i] = myVeh->acc[i+1];
		myVeh->linkID[i] = myVeh->linkID[i+1];
		myVeh->VSP[i] = myVeh->VSP[i+1];
		myVeh->mode[i] = myVeh->mode[i+1];
```

```
		myVeh->tm[i] = myVeh->tm[i+1];
		myVeh->energy[i] = myVeh->energy[i+1];
		myVeh->CO2[i] = myVeh->CO2[i+1];
		myVeh->CO[i] = myVeh->CO[i+1];
		myVeh->HC[i] = myVeh->HC[i+1];
		myVeh->NOx[i] = myVeh->NOx[i+1];
		myVeh->PM[i] = myVeh->PM[i+1];
	}
	myVeh->tm[3] = currTime;
	myVeh->linkID[3] = linkid;
	myVeh->vel[3] = vel_c;
	myVeh->acc[2] = (myVeh->vel[3] - myVeh->vel[1])/2;
	myVeh->VSP[2] = (myVeh->A*myVeh->vel[2] + myVeh->B*pow(myVeh->vel[2],2) +
myVeh->C*pow(myVeh->vel[2],3) + myVeh->M*(myVeh->acc[2] + GRAV*sin(myVeh-
>theta))*myVeh->vel[2])/myVeh->f;
	myVeh->mode[2] = opMode(myVeh->VSP[2], myVeh->vel[2], myVeh->acc[2]);
	if (myVeh->acc[0] < -1 && myVeh->acc[1] < -1 && myVeh->acc[2] < -1) myVeh-
>mode[2] = 0;
	modeBins[myVeh->mode[2]] += 1;
	myVeh->HC[2] = moves[myVeh->sourceType][myVeh->mode[2]][0];
	myVeh->CO[2] = moves[myVeh->sourceType][myVeh->mode[2]][1];
	myVeh->NOx[2] = moves[myVeh->sourceType][myVeh->mode[2]][2];
	myVeh->CO2[2] = moves[myVeh->sourceType][myVeh->mode[2]][3];
	myVeh->energy[2] = moves[myVeh->sourceType][myVeh->mode[2]][4];
	myVeh->PM[2] = moves[myVeh->sourceType][myVeh->mode[2]][5];

	//update overall statistics
	dist_sum += myVeh->vel[2];
	energy_sum += myVeh->energy[2];
	CO2_sum += myVeh->CO2[2];
	CO_sum += myVeh->CO[2];
	HC_sum += myVeh->HC[2];
	NOx_sum += myVeh->NOx[2];
	PM_sum += myVeh->PM[2];

	if (myVeh->EQUIPPED_FLAG == 1) //update statistics for equipped vehicles
	{
		dist_sum_e += myVeh->vel[2];
		energy_sum_e += myVeh->energy[2];
		CO2_sum_e += myVeh->CO2[2];
		CO_sum_e += myVeh->CO[2];
		HC_sum_e += myVeh->HC[2];
		NOx_sum_e += myVeh->NOx[2];
		PM_sum_e += myVeh->PM[2];
	}
	else //EQUIPPED_FLAG == 0  //update statistics for unequipped vehicles
	{
		dist_sum_n += myVeh->vel[2];
		energy_sum_n += myVeh->energy[2];
		CO2_sum_n += myVeh->CO2[2];
		CO_sum_n += myVeh->CO[2];
		HC_sum_n += myVeh->HC[2];
		NOx_sum_n += myVeh->NOx[2];
		PM_sum_n += myVeh->PM[2];
```

```
	}

	//log second-by-second (sbs) data ...seems to cause API crash at the end of
the simulation if the max index is exceeded by congested high volume 30 TS
scenarios
	//timeStamp_sbs[counter_sbs] = (int)(currTime+0.5) - 1;
	//speed_sbs[counter_sbs] = myVeh->vel[2];
	//acc_sbs[counter_sbs] = myVeh->acc[2];
	//VSP_sbs[counter_sbs] = myVeh->VSP[2];
	//vehType_sbs[counter_sbs] = myVeh->sourceType;
	//vehID_sbs[counter_sbs] = myVeh->ID;
	//origin_sbs[counter_sbs] = myVeh->origin;
	//dest_sbs[counter_sbs] = myVeh->dest;
	//counter_sbs++;

	if (myVeh->PARTICIPATION_FLAG == 1) //update statistics for dedicated
lane(s)
	{
		dist_sum_d += myVeh->vel[2];
		energy_sum_d += myVeh->energy[2];
		CO2_sum_d += myVeh->CO2[2];
		CO_sum_d += myVeh->CO[2];
		HC_sum_d += myVeh->HC[2];
		NOx_sum_d += myVeh->NOx[2];
		PM_sum_d += myVeh->PM[2];
	}
	else //myVeh->PARTICIPATION_FLAG == 0 //update statistcs for non-dedicated
lanes
	{
		dist_sum_nd += myVeh->vel[2];
		energy_sum_nd += myVeh->energy[2];
		CO2_sum_nd += myVeh->CO2[2];
		CO_sum_nd += myVeh->CO[2];
		HC_sum_nd += myVeh->HC[2];
		NOx_sum_nd += myVeh->NOx[2];
		PM_sum_nd += myVeh->PM[2];
	}

	//aggregate individual vehicle energy
	myVeh->total_energy += myVeh->energy[2];
	myVeh->total_dist += myVeh->vel[2];
}

float in_qpo_CFM_Speed(LINK* link, VEHICLE* Vp, float CFM_Speed)
{
	float v_lmt = qpg_LNK_speedlimit(link)/2.2369 + 10;
	float currTime = qpg_CFG_simulationTime();
	float recSpeed, currVel = qpg_VHC_speed(Vp);
	int linkid = qpg_LNK_index(link), mode, currZone = qpg_LNK_zone(link);
	float timeStep = qpg_CFG_timeStep();
	float currentClearance = 0, currentSpacing = 0, targetDistance = 0,
relativeSpeed = 0;
	//VEHICLE* ahead = qpg_VHC_ahead(Vp); //pointer to vehicle directly ahead
(on the same link)
```

```
	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp); // members of
current vehicle (pointer is from input parameter)
	VEHICLE* ahead = myVeh->vehicleAhead; //qpg_VHC_ahead(Vp); //pointer to
vehicle directly ahead (on the same link)
	VEHICLE* vehicleDirectlyAhead = qpg_VHC_ahead(Vp); //pointer to vehicle
directly ahead (on the same link), NULL if no vehicle is ahead on same link and
lane
	VEHICLE_DATA *vehAhead; //=
(VEHICLE_DATA*)qpg_VHC_userdata(vehicleDirectlyAhead);		// =
(VEHICLE_DATA*)qpg_VHC_userdata(ahead); // members of vehicle directly ahead,
first check if ahead is not null
	int currentLane = qpg_VHC_lane(Vp);
	int linkTransition = 0; // DEBUG only
	float acceleration = 0; // used for acceleration color scheme
	float colorScale = 0;
	float x1 = 0;
	float y1 = 0;
	float z1 = 0; // unused
	float b1 = 0; // unused
	float g1 = 0; // unused
	float x2 = 0;
	float y2 = 0;
	float z2 = 0; // unused
	float b2 = 0; // unused
	float g2 = 0; // unused

	VEHICLE* laneLinkLeaderP; //used for determining if merging behavior is
necessary
	VEHICLE* laneLinkRearP; //used for determining if merging behavior is
necessary/possible
	VEHICLE* testVp;
	VEHICLE* nearestAdjacentBehindVp;
	VEHICLE* nearestAdjacentAheadVp;
	VEHICLE* behindNearestAdjacentBehindVp; // vehicle pointer to vehicle
behind nearestAdjacentBehindVp
	VEHICLE_DATA *nearestAdjacentBehindVdata;
	VEHICLE_DATA *nearestAdjacentAheadVdata;
	VEHICLE_DATA *behindNearestAdjacentBehindVdata; // vehicle data of vehicle
behind nearestAdjacentBehindVp
	float adjacentTargetDistance = 0; //target distance between the current
vehicle and the nearest adjacent vehicle ahead in the target lane
	float adjacentTargetDistanceBehind = 0; //target distance (difference
between spacing and target) of nearest vehicle adjacent and behind the current
vehicle relative to the current vehicle
	float currentClearance2 = 0;
	float currentSpacing2 = 0;

	int condition = 0; //use as bool, 0 = false, 1 = true

	//Acceleration/Deceleration Constraint Variables
	float deltaSpeed = 0;

	//Splitting variables
```

```
	VEHICLE* vehicleDirectlyBehind; //qpg_VHC_behind(Vp); //pointer to vehicle
directly behind (on the same link-lane), NULL if no vehicle is behind on same link
and lane
	VEHICLE_DATA *BehindVdata;
	//VEHICLE* vehicleBehindBehind = qpg_VHC_behind(qpg_VHC_behind(Vp));
	//VEHICLE_DATA *BehindBehindVdata;
	float currentRearClearance = 0; //clearance of vehicle behind to current
vehicle (distance from front bumper of rear vehicle to front bumper of the current
vehicle)
	float x3 = 0;
	float y3 = 0;
	float z3 = 0; // unused
	float b3 = 0; // unused
	float g3 = 0; // unused
	float x4 = 0;
	float y4 = 0;
	float z4 = 0; // unused
	float b4 = 0; // unused
	float g4 = 0; // unused

	//Dedicated Lane variables
	int numberOfLanes = qpg_LNK_lanes(link); // number of lanes on the current
link

	//test
	LINK *nextLink;
	int nextLinkIndex;

	//node position variables, used for triggering splitting protocol and/or
setting participation flag to 0
	/*float x5 = 0;
	float y5 = 0;
	float z5 = 0;*/
	float distToDestNode = 100000000; // distance to destination node,
initialized to large number since we are using a max threshold comparison

	//Update color of vehicle if equipped (speed color scheme)
	//if (myVeh->EQUIPPED_FLAG == 1)
	//{
	//	colorScale = (MAX(MIN(qpg_VHC_speed(Vp),32),17)-17)/15;
	//	qps_DRW_vehicleColour(Vp, qpg_DRW_colourScale(colorScale));
//colourScale, red is 1, blue is 0, other colors are in between too
	//}

	////Change color of vehicle if equipped and not participating
	//if (myVeh->EQUIPPED_FLAG == 1 && myVeh->PARTICIPATION_FLAG == 0 && myVeh-
>PRIOR_PARTICIPATION_FLAG == 1) //if flag switches from 1 to 0
	//{
	//	qps_DRW_vehicleColour(Vp, 4294967040); //cyan 4294967040
(0xFFFFFF00) //yellow 4278255615 (0xFF00FFFF) //blue 4294901760 (OxFFFF0000)
	//}
	////update prior participation flag for the next time step
	//myVeh->PRIOR_PARTICIPATION_FLAG = myVeh->PARTICIPATION_FLAG;
```

```
	//Change color of vehicle if equipped and leaving for exit
	if (myVeh->EQUIPPED_FLAG == 1 && myVeh->EXIT_APPROACHING_FLAG == 1)
	{
		qps_DRW_vehicleColour(Vp, 4294967040); //cyan 4294967040
(0xFFFFFF00) //yellow 4278255615 (0xFF00FFFF) //blue 4294901760 (OxFFFF0000)
	}


	////Set participation flag of vehicles to 1 after merging from on-ramp with
mainstream traffic (NOTE: this section is network-unique) ...generalize later
	//if (myVeh->MAINSTREAM_FLAG == 0)
	//{
	//	if (myVeh->origin == 4)
	//	{
	//		/*qps_GUI_printf("Vehicle %d trip %d -> %d on link %s; Case =
%s, Participation Flag = %d \n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "Zone = 4", myVeh-
>PARTICIPATION_FLAG);
	//		qps_GUI_simRunning(0);*/
	//		if (myVeh->PARTICIPATION_FLAG == 0 && myVeh->splitting == 0)
	//		{
	//			/*qps_GUI_printf("Vehicle %d trip %d -> %d on link %s;
Case = %s, Participation Flag = %d \n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "Zone = 4", myVeh-
>PARTICIPATION_FLAG);
	//			qps_GUI_simRunning(0);*/
	//			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1,
&b1, &g1);
	//			//qps_GUI_printf("Vehicle %d trip %d -> %d on link %s;
Case = %s, X position = %f \n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "Zone = 4", x1);
	//			//qps_GUI_simRunning(0);
	//			x1 = -x1 - 310.776192 - 3116.94576;
	//			//assert(x1 > -1000);
	//			if (x1 > (-789+200)) //meters
	//			{
	//				myVeh->PARTICIPATION_FLAG = 1;
	//				myVeh->MAINSTREAM_FLAG = 1; //only set the
participation flag to 1 once
	//				//qps_GUI_printf("Vehicle %d trip %d -> %d on
link %s; Case = %s\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "Zone 4");
	//				//qps_GUI_simRunning(0);
	//			}
	//		}
	//	}
	//	else if (myVeh->origin == 6)
	//	{
	//		if (myVeh->PARTICIPATION_FLAG == 0 && myVeh->splitting == 0)
	//		{
	//			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1,
&b1, &g1);
	//			x1 = -x1 - 310.776192 - 3116.94576;
	//			if (x1 > (1223+200))
```

```
	//		{
	//			myVeh->PARTICIPATION_FLAG = 1;
	//			myVeh->MAINSTREAM_FLAG = 1; //only set the
participation flag to 1 once
	//		}
	//		}
	//	}
	//}

	//comment out the previous section, and set Participation flag based on
dedicatedLanes parameter and number of lanes (on current link) and current lane
	if (myVeh->PARTICIPATION_FLAG == 0 && myVeh->splitting == 0 && myVeh-
>EXIT_APPROACHING_FLAG == 0)
	{
		if (numberOfLanes > 2) // if vehicle is on a mainstream link (i.e.
not a freeway exit)
		{
			if (numberOfLanes - currentLane + 1 <= dedicatedLanes)
			{
				if (strcmp(qpg_LNK_name(link), "4:218") == 0 ||
strcmp(qpg_LNK_name(link), "218:5") == 0 || strcmp(qpg_LNK_name(link), "5:3") ==
0) //probihited links for platoon formation, ADDED for v5_1
				{
					myVeh->PARTICIPATION_FLAG = 0; //vehicle is near
a confluence section, therefore, do not form platoon (this line is technically
unnecessary)
				}
				else
				{
					myVeh->PARTICIPATION_FLAG = 1; //vehicle is in
one of the dedicated lanes
					myVeh->PARTICIPATED_FLAG = 1; //permanently
switch PARTICIPATED_FLAG to 1
					counter_participated += 1;
				}
			}
		}
	}
	else if (myVeh->PARTICIPATION_FLAG == 1)
	{
		if (numberOfLanes > 2) // if vehicle is on a mainstream link
		{
			if (numberOfLanes - currentLane + 1 > dedicatedLanes)
			{
				myVeh->PARTICIPATION_FLAG = 0; //vehicle is NOT on one
of the dedicated lanes
			}
		}
		else // vehicle is not on a mainstream link
		{
			myVeh->PARTICIPATION_FLAG = 0;
		}

	}
```

```
	if (link != myVeh->currentLink) //update currentLink
	{
		linkTransition = 1;
		myVeh->currentLink = link;
	}
	// update platoon lane index during transition to next link (this assumed
that vehicles were not changing lanes!...update platoonLaneIndex during platoon
formation)
	/*if (link != myVeh->currentLink)
	{
		linkTransition = 1;*/
		//myVeh->currentLink = link;

		// Check next link geometry to appropriately modify platoonLaneIndex
		// 1) number of lanes drops (assume drop of only one lane)
		if (qpg_LNK_lanes(qpg_VHC_link(Vp)) ==
qpg_LNK_lanes(qpg_LNK_nearside(qpg_VHC_link(Vp)))+1) //does nearside always return
the next link?...is nearside length correct?
		{
			myVeh->laneDropApproaching = 1;
			//qps_GUI_printf("laneDropApproaching = %d ", myVeh-
>laneDropApproaching);
			if (myVeh->departLaneIndex == 1)
			{
				myVeh->platoonLaneIndex = 2;
				//qps_GUI_printf("platoon Lane Index is: %d ", myVeh-
>platoonLaneIndex);
			}
		}
		else
		{
			myVeh->laneDropApproaching = 0;

			//qps_GUI_printf("laneDropApproaching = %d ", myVeh-
>laneDropApproaching);
			//qps_GUI_printf("platoon Lane Index was: %d ", myVeh-
>platoonLaneIndex);
			// update platoon lane index during transition to next link
			myVeh->platoonLaneIndex = qpg_VHC_lane(Vp); //current lane is
the preffered lane on new link //qpg_VHC_nextLane(Vp); // check if this is
accurate (nextLane is incorrect, since vehicles have already reached the next
link)
			//qps_GUI_printf("platoon Lane Index is: %d ", myVeh-
>platoonLaneIndex);
		}
	//}

	////// Restricted lane change version 3
	//currentLane = qpg_VHC_lane(Vp);
	//if (myVeh->laneDropApproaching == 1) //the only case in which the lane
restriction is not enforced is when departLaneIndex == 1 and currentLane == 1
	//{
```

```
	//	if (myVeh->departLaneIndex == 1 && currentLane == 2) // once vehicle
on bottom lane (lane 1) changes lanes to lane 2, keep the vehicle on lane 2
	//	{
	//		qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex); //platoonLaneIndex should equal 2 in this case
	//	}
	//	else if (myVeh->departLaneIndex != 1) // else prevent lane changes
	//	{
	//		qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex); //platoonLaneIndex should equal 2+ in this case
	//	}
	//}
	//else // no lane drop approaching
	//{
	//	qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex);
	//}

	//// Restricted lane change version 4
	//currentLane = qpg_VHC_lane(Vp); // moved this initialization to the
declaration above (this should work)
	if (myVeh->EQUIPPED_FLAG == 1 && myVeh->PARTICIPATION_FLAG == 1 && (myVeh-
>CACCstatus == -1 || myVeh->CACCstatus == 1)) // only restrict lane changes for
equipped vehicles in platoons who are participating
	{
		if (myVeh->laneDropApproaching == 1) //the only case in which the
lane restriction is not enforced is when departLaneIndex == 1 and currentLane == 1
		{
			if (myVeh->departLaneIndex == 1 && currentLane == 2) // once
vehicle on bottom lane (lane 1) changes lanes to lane 2, keep the vehicle on lane
2
			{
				qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex); //platoonLaneIndex should equal 2 in this case
			}
			else if (myVeh->departLaneIndex != 1) // else prevent lane
changes
			{
				qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex); //platoonLaneIndex should equal 2+ in this case
			}
		}
		else // no lane drop approaching
		{
			qps_VHC_laneRange(Vp, myVeh->platoonLaneIndex, myVeh-
>platoonLaneIndex);
		}
	} //else laneRange is determined by Paramics (vehicles are free to change
lanes as they wish)

	//Set vehicle Behind, if it exists
	if (Vp != qpg_LNK_vehicleTail(link, currentLane))
	{
		vehicleDirectlyBehind = qpg_VHC_behind(Vp);
```

```
	}
	else
	{
		vehicleDirectlyBehind = NULL;
	}

	// calculate emissions
	if (fabs(currTime - (int)(currTime+0.5)) > DELTA || currZone == myVeh-
>origin || currZone == myVeh->dest) //these are conditions under which emissions
should not be calculated
		goto CACC;
	if (myVeh->first2sec < 3)
	{
		myVeh->tm[myVeh->first2sec] = currTime;
		myVeh->linkID[myVeh->first2sec] = linkid;
		myVeh->vel[myVeh->first2sec] = currVel;
		myVeh->first2sec++;
		goto CACC;
	}
	calEmissions(myVeh, currTime, currVel, linkid); //emissions are calculated
every three seconds

CACC:
	//Corrected state machine 2
	//1A) Update current vehicle's vehicle directly ahead parameter
	if (vehicleDirectlyAhead != NULL) // the vehicle ahead is on the same link
	{
		ahead = vehicleDirectlyAhead;
		myVeh->vehicleAhead = ahead; //initially NULL, should remain set for
duration of simulation...but can be changed if the vehicle directly ahead changes
lanes
		vehAhead = (VEHICLE_DATA*)qpg_VHC_userdata(ahead); // members of
vehicle directly ahead
	}
	else if (myVeh->vehicleAhead != NULL) // vehicleAhead was already set when
the two vehicles were on the same link, now that the vehicleAhead is on the next
link, use vehicleAhead for vehAhead data
	{
		/*if (myVeh->vehicleAhead == 0)
		{
			qps_GUI_printf("ERROR: Current Vehicle ID = %d ",
qpg_VHC_uniqueID(Vp));
		}*/
		if (link == qpg_VHC_link(myVeh->vehicleAhead)) //check to see if the
vehicleAhead is on the same link
		{
			if (currentLane == qpg_VHC_lane(myVeh->vehicleAhead)) //check
to see if the vehicleAhead is on the same lane as the current vehicle
			{
				vehAhead = (VEHICLE_DATA*)qpg_VHC_userdata(myVeh-
>vehicleAhead);
			}
			else //either the vehicleAhead changed lanes, or the current
vehicle changed lanes...reset vehicleAhead field (there is no vehicle ahead)
```

```
                        {
                                myVeh->vehicleAhead = NULL;
                                vehAhead = NULL;
                        }
                }
                else //we assume that the vehicleAhead remained in the same lane
after transitioning to the next link
                {
                        vehAhead = (VEHICLE_DATA*)qpg_VHC_userdata(myVeh-
>vehicleAhead);
                }
        }
        //1B) Update current vehicle's vehicle directly behind parameter
        if (vehicleDirectlyBehind != NULL) //the vehicle behind is on the same link
        {
                myVeh->vehicleBehindP = vehicleDirectlyBehind;
                BehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(vehicleDirectlyBehind);
        }
        else if (myVeh->vehicleBehindP != NULL) // vehicleBehindP was already set
when the two vehicles were on the same link, now that the current vehicle is on
the next link, use myVeh->vehicleBehindP for BehindVdata
        {
                if (link == qpg_VHC_link(myVeh->vehicleBehindP)) //check to see if
the vehicleBehind is on the same link
                {
                        if (currentLane == qpg_VHC_lane(myVeh->vehicleBehindP))
//check to see if the vehicleBehind is on the same lane as the current vehicle
                        {
                                BehindVdata = (VEHICLE_DATA*)qpg_VHC_userdata(myVeh-
>vehicleBehindP); //this should never be executed (statement 8 lines earlier
should set this case)
                        }
                        else //either the vehicleBehind changed lanes, or the current
vehicle changed lanes...reset vehicleBehind field (there is no vehicle behind)
                        {
                                myVeh->vehicleBehindP = NULL;
                                BehindVdata = NULL;
                        }
                }
                else //we assume that the vehicleBehind remained in the same lane
after transitioning to the next link
                {
                        BehindVdata = (VEHICLE_DATA*)qpg_VHC_userdata(myVeh-
>vehicleBehindP);
                }
        }
        else
        {
                BehindVdata = NULL;
        }

        //2) Check if a vehicle needs to make an immediate lane change due to close
proximity to lane drop
```

```
	if (myVeh->merging == 0 && currentLane == 1 && myVeh->laneDropApproaching
== 1 && (qpg_VHC_distance(Vp) < 60.96)) //60.96 meters = 200 ft, this is range for
imminent lane change check
	{

		//Find the adjacent vehicle behind the current vehicle
		if (qpg_LNK_vehicles(link, 2) >= 2) //check if there are at least
two vehicles on the adjacent lane, (at this point we are assuming that the
adjacent lane is lane 2) ...minimum of 2 vehicles is considered necessary since
there must be at least 2 vehicles in the lane for a leader to exist
		{
			//qps_GUI_printf("entered 1_1");
			laneLinkLeaderP = qpg_LNK_vehicleHead(link, 2); // vehicle at
the head of the link on lane 2
			laneLinkRearP = qpg_LNK_vehicleTail(link, 2); // last vehicle
on lane 2 of the link

			if (qpg_VHC_distance(laneLinkRearP) > qpg_VHC_distance(Vp))
//make sure that there is at least one vehicle in the adjacent lane which is
behind the current vehicle
			{
				//Find the nearestAdjacentBehind vehicle
				//testVp = qpg_VHC_behind(laneLinkLeaderP);
				//while (qpg_VHC_distance(testVp) <
qpg_VHC_distance(Vp)) //make sure this terminates!
				//{
				//	testVp = qpg_VHC_behind(testVp); //iterate
backwards on lane
				//}
				//nearestAdjacentBehindVp = testVp;
				//nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);

				//if ((qpg_VHC_distance(laneLinkLeaderP) +
vehicleLength) > qpg_VHC_distance(Vp)) //make sure there are no other vehicles
ahead of the current vehicle in the adjacent lane (this will ensure that the
current vehicle will not be accelerating into a vehicle ahead on the adjacent
lane)
				if ((qpg_VHC_distance(laneLinkLeaderP) +
1.5*vehicleLength) > qpg_VHC_distance(Vp)) //if the current vehicle is less than a
car length behind the adjacent leader vehicle, then accelerate the current vehicle
(whether equipped or not)
				{
					nearestAdjacentBehindVp = laneLinkLeaderP;
					nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);
					if (nearestAdjacentBehindVdata->CACCstatus == 1
&& (qpg_VHC_distance(nearestAdjacentBehindVp) < 60.96)) //check for close leader
vehicle
					{
						//speed current vehicle up
						return (qpg_VHC_speed(Vp) +
0.9*max_accel_mpss*timeStep); //check to see if vehicle accelerates into a vehicle
ahead
```

```
                                }
                        }
                }
        }
}

        //3) Update current vehicle's CACC state, Update vehicle directly ahead's
CACC state (if necessary), determine if manuevers are necessary
        if (myVeh->EQUIPPED_FLAG == 0) //uneqipped vehicle logic
        {
                if (myVeh->merging == 0 && currentLane == 1 && myVeh-
>laneDropApproaching == 1 && (qpg_VHC_distance(Vp) < (2*304.8))) //2x ... 304.8
meters = 1000 ft, determine if the platoon split scenario is necessary to
accomodate unequipped vehicle changing lanes near lane drop
                {
                        //determine if a platoon occupies the adjacent target lane
                        //Method 2: use the head vehicle on the link, and iterate
backwards using the vehicle behind function until a vehicle behind the current
vehicle is found
                        if (qpg_LNK_vehicles(link, 2) >= 2) //check if there are at
least two vehicles on the adjacent lane, (at this point we are assuming that the
adjacent lane is lane 2)
                        {
                                //qps_GUI_printf("entered 1_1");
                                laneLinkLeaderP = qpg_LNK_vehicleHead(link, 2); //
vehicle at the head of the link on lane 2
                                laneLinkRearP = qpg_LNK_vehicleTail(link, 2); // last
vehicle on lane 2 of the link

                                if (qpg_VHC_distance(laneLinkLeaderP) <
qpg_VHC_distance(Vp) && qpg_VHC_distance(Vp) < qpg_VHC_distance(laneLinkRearP))
//check if there is a vehicle ahead on the adjacent lane, also check if there is
vehicle behind on the adjacent lane
                                {
                                        //qps_GUI_printf("entered 1_2");
                                        testVp = qpg_VHC_behind(laneLinkLeaderP);
                                        while (qpg_VHC_distance(testVp) <
qpg_VHC_distance(Vp)) //make sure this terminates!
                                        {
                                                //qps_GUI_printf("entered 1_3");
                                                testVp = qpg_VHC_behind(testVp);
//iterate backwards on lane
                                        }
                                        nearestAdjacentBehindVp = testVp;
                                        nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);
                                        nearestAdjacentAheadVp = qpg_VHC_ahead(testVp);
                                        nearestAdjacentAheadVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentAheadVp);

                                        //qps_GUI_printf("Current Vehicle ID = %d ",
qpg_VHC_uniqueID(Vp));
                                        //qps_GUI_printf("Adjacent Behind ID = %d ",
qpg_VHC_uniqueID(nearestAdjacentBehindVp));
```

```
                                    //qps_GUI_printf("Adjacent Ahead ID = %d ",
qpg_VHC_uniqueID(nearestAdjacentAheadVp));

                                    behindNearestAdjacentBehindVp =
qpg_VHC_behind(nearestAdjacentBehindVp);
                                    if (behindNearestAdjacentBehindVp != 0 &&
behindNearestAdjacentBehindVp != NULL)
                                    {
                                            behindNearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(behindNearestAdjacentBehindVp);
                                            if (behindNearestAdjacentBehindVdata-
>CACCstatus == -1)
                                            {
                                                    condition = 1;
                                            }
                                            else
                                            {
                                                    condition = 0;
                                            }
                                    }
                                    else
                                    {
                                            condition = 0;
                                    }

                                    //ADDED LOGIC to uneqipped case: vehicle behind
adjacent behind vehicle cannot be a follower (since splitting near the tail of a
platoon would take the tail of the platoon out of a platoon altogether)
                                    //if (condition == 0) //perhaps slow down
vehicle?
                                    if (condition == 1 &&
nearestAdjacentBehindVdata->CACCstatus == -1 && (nearestAdjacentAheadVdata-
>CACCstatus == -1 || (nearestAdjacentAheadVdata->CACCstatus == 1 && condition ==
1) )) //if both of the vehicles on the adjacent lanes are followers ...or if
vehicle ahead is a leader of a platoon of length >= 3
                                    {
                                            //qps_GUI_printf("entered 0_4");
                                            myVeh->vehicleAdjacentBehindP =
nearestAdjacentBehindVp; //keep track of nearest vehicle in adjacent target lane
behind current vehicle
                                            myVeh->vehicleAdjacentAheadP =
nearestAdjacentAheadVp; //also keep track of nearest vehicle in adjacent target
lane ahead of the current vehicle (in case in changes links before the manuever is
completed)...might need to check if another vehicle cuts in ahead of the current
vehicle (thereby changing the nearest Adjacent vehicle ahead)
                                            ////Set current vehicle to follower
(check if this causes unforeseen errors)
                                            //myVeh->CACCstatus = -1; // current
vehicle is set to be a follower
                                            myVeh->merging = 1; //Initiate Stage 1 of
merging (longitudinal relative positioning)

                                            //nearestAdjacentBehindVdata-
>vehTargetSpacing = 4*targetSpacing + vehicleLength; //this is more than the
```

```
minimum spacing necessary, paramics should allow the vehicle to change lane in
between the two split platoons
                                        nearestAdjacentBehindVdata->CACCstatus =
1; //vehicle becomes a leader at the beginning of the platoon split, and should
slow down automatically
                                        nearestAdjacentBehindVdata->splitting =
1; //set splitting flag to prevent reformation of the platoon


                                        ////check if spacing has been achieved by
nearestAdjacentBehindVdata
                                        ////currentClearance2 =
qpg_VHC_distance(nearestAdjacentBehindVp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is on the lane adjacent to the
current vehicle
                                        ////currentSpacing2 = currentClearance2 -
vehicleLength;
                                        ////adjacentTargetDistance =
currentSpacing2 - nearestAdjacentBehindVdata->vehTargetSpacing; // changed from
targetSpacing

                                        ////check spacing between
nearestAdjacentBehind and current vehicle
                                        //currentClearance2 =
qpg_VHC_distance(nearestAdjacentBehindVp) - qpg_VHC_distance(Vp); //first vehicle
is on the lane adjacent to the current vehicle
                                        //currentSpacing2 = currentClearance2 -
vehicleLength;
                                        //adjacentTargetDistanceBehind =
currentSpacing2 - myVeh->vehTargetSpacing; // vehTargetSpacing should equal
default (5)


                                        ////Apply altered follower code to the
current vehicle
                                        //if (link ==
qpg_VHC_link(nearestAdjacentAheadVp))
                                        //{
                                        //     currentClearance =
qpg_VHC_distance(Vp) - qpg_VHC_distance(nearestAdjacentAheadVp); //this is
actually cross-lane clearance, check for inaccuracies in this line
                                        //}
                                        //else //nearestAdjacentAhead vehicle has
changed links
                                        //{
                                        //     //Method 2 of calculating vehicle
clearance when vehicles are on consecutive links
                                        //     qpg_POS_vehicle(Vp,
qpg_VHC_link(Vp), &x1, &y1, &z1, &b1, &g1);
                                        //
       qpg_POS_vehicle(nearestAdjacentAheadVp,
qpg_VHC_link(nearestAdjacentAheadVp), &x2, &y2, &z2, &b2, &g2);
                                        //     currentClearance = sqrt((x2-
x1)*(x2-x1) + (y2-y1)*(y2-y1)); // assumes both vehicles are the same type
```

```
                                        //}
                                        //currentSpacing = currentClearance -
vehicleLength;
                                        //targetDistance = currentSpacing -
myVeh->vehTargetSpacing; // changed from targetSpacing
                                        //if (targetDistance > 0)
                                        //{
                                        //     relativeSpeed = targetDistance/2;
                                        //}
                                        //else
                                        //{
                                        //     relativeSpeed = targetDistance/2;
// 0.5
                                        //}

                                        ////if ((abs(targetDistance) < 1) &&
(adjacentTargetDistance > -1)) //if current vehicle is close to the desired
control point, and the platoon has finished splitting, then proceed with merging
operation
                                        //if ((targetDistance > -1) &&
(adjacentTargetDistanceBehind > -1) && ((qpg_VHC_speed(Vp) -
qpg_VHC_speed(nearestAdjacentBehindVp)) > -2.2352)) // if the current vehicle is
sufficiently behind the adjacent ahead vehicle, and if the current vehicle is
sufficiently ahead of the adjacent behind vehicle, and the current vehicle speed
is within 5 mph of the adjacent behind vehicle speed
                                        //{
                                        //     //qps_GUI_printf("entered 0_5");
                                        //     //maybe set let in flag of
nearestAdjacentBehind?
                                        //     //Method 1: do gradual lane change
using laneRanges
                                        //     qps_VHC_laneRange(Vp, 2, 2);
//check to see if this works
                                        //     myVeh->merging = 2;
                                        //}
                                        //else
                                        //{
                                        //     //qps_GUI_printf("entered 0_6");
                                        //     qps_VHC_laneRange(Vp, 1, 1);
//prevent vehicle from making premature lane change
                                        //}
                                        ////qps_GUI_simRunning(0); // works! 0 =
false

                                        ////Apply Acceleration and Deceleration
Constraints
                                        //deltaSpeed =
(qpg_VHC_speed(nearestAdjacentAheadVp) + relativeSpeed) - qpg_VHC_speed(Vp);
                                        //if (deltaSpeed >
max_accel_mpss*timeStep)
                                        //{
                                        //     return MAX((qpg_VHC_speed(Vp) +
max_accel_mpss*timeStep), 0);
                                        //}
```

```
                                        //else if (deltaSpeed <
min_decel_mpss*timeStep)
                                        //{
                                        //     return MAX((qpg_VHC_speed(Vp) +
min_decel_mpss*timeStep), 0);
                                        //}
                                        //else
                                        //{
                                        //     return
MAX((qpg_VHC_speed(nearestAdjacentAheadVp) + relativeSpeed), 0); //uses speed of
vehicle in adjacent lane
                                        //}

                                        return CFM_Speed; //maybe look at
altering this speed, if the manuevers are not being completed
                                    }

                            }
                        }
                }
                else
                {
                        return CFM_Speed;
                }
        }
        if (myVeh->merging == 0 && currentLane == 1 && myVeh->laneDropApproaching
== 1 && (qpg_VHC_distance(Vp) < (2*304.8))) //2x ... 304.8 meters = 1000 ft,
determine if the lane merge scenario is necessary to bottleneck and adjacent
platoon
        {
                //qps_GUI_printf("entered 1");
                //qps_GUI_simRunning(0); // works! 0 = false
                //determine if a platoon occupies the adjacent target lane
                //Method 1: use array of vehicles on adjacent target lane, and
iterate through array to find the Nearest "Follower" ahead and behind (if they
exist)
                //qpg_LNK_vehicleList(link, 2, qpg_LNK_vehicles(link, 2), VEHICLE*
vehicle[]); //LINK* link, int lane, int count, VEHICLE* vehicle[] ...//return
array of vehicle pointers of vehicles on a certain lane of a link ...is count a
return value, or an input?

                //Method 2: use the head vehicle on the link, and iterate backwards
using the vehicle behind function until a vehicle behind the current vehicle is
found
                if (qpg_LNK_vehicles(link, 2) >= 2) //check if there are at least
two vehicles on the adjacent lane, (at this point we are assuming that the
adjacent lane is lane 2)
                {
                        //qps_GUI_printf("entered 1_1");
                        laneLinkLeaderP = qpg_LNK_vehicleHead(link, 2); // vehicle at
the head of the link on lane 2
                        laneLinkRearP = qpg_LNK_vehicleTail(link, 2); // last vehicle
on lane 2 of the link
```

```
                    if (qpg_VHC_distance(laneLinkLeaderP) < qpg_VHC_distance(Vp)
&& qpg_VHC_distance(Vp) < qpg_VHC_distance(laneLinkRearP)) //check if there is a
vehicle ahead on the adjacent lane, also check if there is vehicle behind on the
adjacent lane
                    {
                           //qps_GUI_printf("entered 1_2");
                           testVp = qpg_VHC_behind(laneLinkLeaderP);
                           while (qpg_VHC_distance(testVp) < qpg_VHC_distance(Vp))
//make sure this terminates!
                           {
                                  //qps_GUI_printf("entered 1_3");
                                  testVp = qpg_VHC_behind(testVp); //iterate
backwards on lane
                           }
                           nearestAdjacentBehindVp = testVp;
                           nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);
                           nearestAdjacentAheadVp = qpg_VHC_ahead(testVp);
                           nearestAdjacentAheadVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentAheadVp);

                           //qps_GUI_printf("Current Vehilce ID = %d ",
qpg_VHC_uniqueID(Vp));
                           //qps_GUI_printf("Adjacent Behind ID = %d ",
qpg_VHC_uniqueID(nearestAdjacentBehindVp));
                           //qps_GUI_printf("Adjacent Ahead ID = %d ",
qpg_VHC_uniqueID(nearestAdjacentAheadVp));

                           behindNearestAdjacentBehindVp =
qpg_VHC_behind(nearestAdjacentBehindVp);
                           if (behindNearestAdjacentBehindVp != 0 &&
behindNearestAdjacentBehindVp != NULL)
                           {
                                  behindNearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(behindNearestAdjacentBehindVp);
                                  if (behindNearestAdjacentBehindVdata->CACCstatus
== -1)
                                  {
                                         condition = 1;
                                  }
                                  else
                                  {
                                         condition = 0;
                                  }
                           }
                           else
                           {
                                  condition = 0;
                           }

                           if (nearestAdjacentBehindVdata->CACCstatus == -1 &&
(nearestAdjacentAheadVdata->CACCstatus == -1 || (nearestAdjacentAheadVdata-
>CACCstatus == 1 && condition == 1) )) //if both of the vehicles on the adjacent
lanes are followers ...or if vehicle ahead is a leader of a platoon of length >= 3
```

```
                        {
                                //qps_GUI_printf("entered 0_4");
                                myVeh->vehicleAdjacentBehindP =
nearestAdjacentBehindVp; //keep track of nearest vehicle in adjacent target lane
behind current vehicle
                                myVeh->vehicleAdjacentAheadP =
nearestAdjacentAheadVp; //also keep track of nearest vehicle in adjacent target
lane ahead of the current vehicle (in case in changes links before the manuever is
completed)...might need to check if another vehicle cuts in ahead of the current
vehicle (thereby changing the nearest Adjacent vehicle ahead)
                                //Set current vehicle to follower (check if this
causes unforeseen errors)
                                myVeh->CACCstatus = -1; // current vehicle is
set to be a follower
                                myVeh->merging = 1; //Initiate Stage 1 of
merging (longitudinal relative positioning)


                                //nearestAdjacentBehindVdata->vehTargetSpacing =
2*targetSpacing + vehicleLength; //this is the minimum spacing necessary
                                nearestAdjacentBehindVdata->vehTargetSpacing +=
targetSpacing + vehicleLength; //increment approach

                                if (vehicleDirectlyAhead != NULL) // check for
imminent collision with vehicle ahead (conflict between merging control logic and
car following logic)
                                {
                                        //if the vehicle ahead is too close, or
the vehicle ahead is behind the bumper of the leader, then reset targetSpacing for
current vehicle and adjacent Behind vehicle
                                        if (((qpg_VHC_distance(Vp) -
qpg_VHC_distance(vehicleDirectlyAhead)) < 4) ||
((qpg_VHC_distance(vehicleDirectlyAhead) - (qpg_VHC_distance(myVeh-
>vehicleAdjacentAheadP) + vehicleLength)) > 0))
                                        {
                                                myVeh->vehTargetSpacing =
2*targetSpacing + vehicleLength; // same as incrementing by targetSpacing +
vehicleLength
                                                nearestAdjacentBehindVdata-
>vehTargetSpacing = 3*targetSpacing + 2*vehicleLength; // same as incrementing by
targetSpacing + vehicleLength
                                        }
                                }


                                //check if spacing has been achieved by
nearestAdjacentBehindVdata
                                //currentClearance2 =
qpg_VHC_distance(nearestAdjacentBehindVp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is on the lane adjacent to the
current vehicle
                                //currentSpacing2 = currentClearance2 -
vehicleLength;
```

```
                                //adjacentTargetDistance = currentSpacing2 -
nearestAdjacentBehindVdata->vehTargetSpacing; // changed from targetSpacing

                                //check spacing between nearestAdjacentBehind
and current vehicle
                                currentClearance2 =
qpg_VHC_distance(nearestAdjacentBehindVp) - qpg_VHC_distance(Vp); //first vehicle
is on the lane adjacent to the current vehicle
                                currentSpacing2 = currentClearance2 -
vehicleLength;
                                adjacentTargetDistanceBehind = currentSpacing2 -
targetSpacing; //myVeh->vehTargetSpacing; // vehTargetSpacing should equal default
(5)


                                //Apply altered follower code to the current
vehicle
                                if (link ==
qpg_VHC_link(nearestAdjacentAheadVp))
                                {
                                        currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is actually cross-lane clearance,
check for inaccuracies in this line
                                }
                                else //nearestAdjacentAhead vehicle has changed
links
                                {
                                        //Method 2 of calculating vehicle
clearance when vehicles are on consecutive links
                                        qpg_POS_vehicle(Vp, qpg_VHC_link(Vp),
&x1, &y1, &z1, &b1, &g1);
                                        qpg_POS_vehicle(nearestAdjacentAheadVp,
qpg_VHC_link(nearestAdjacentAheadVp), &x2, &y2, &z2, &b2, &g2);
                                        currentClearance = sqrt((x2-x1)*(x2-x1) +
(y2-y1)*(y2-y1)); // assumes both vehicles are the same type
                                }
                                currentSpacing = currentClearance -
vehicleLength;
                                targetDistance = currentSpacing - myVeh-
>vehTargetSpacing; // changed from targetSpacing
                                if (targetDistance > 0)
                                {
                                        relativeSpeed = targetDistance/2;
                                }
                                else
                                {
                                        relativeSpeed = targetDistance/2; // 0.5
                                }

                                //if ((abs(targetDistance) < 1) &&
(adjacentTargetDistance > -1)) //if current vehicle is close to the desired
control point, and the platoon has finished splitting, then proceed with merging
operation
```

```
                                if ((targetDistance > -1) &&
(adjacentTargetDistanceBehind > -1) && ((qpg_VHC_speed(Vp) -
qpg_VHC_speed(nearestAdjacentBehindVp)) > -2.2352)) // if the current vehicle is
sufficiently behind the adjacent ahead vehicle, and if the current vehicle is
sufficiently ahead of the adjacent behind vehicle, and the current vehicle speed
is within 5 mph of the adjacent behind vehicle speed
                                {
                                        //qps_GUI_printf("entered 0_5");
                                        //maybe set let in flag of
nearestAdjacentBehind?
                                        //Method 1: do gradual lane change using
laneRanges
                                        qps_VHC_laneRange(Vp, 2, 2); //check to
see if this works
                                        myVeh->merging = 2;
                                }
                                else
                                {
                                        //qps_GUI_printf("entered 0_6");
                                        qps_VHC_laneRange(Vp, 1, 1); //prevent
vehicle from making premature lane change
                                }
                                //qps_GUI_simRunning(0); // works! 0 = false

                                //Apply Acceleration and Deceleration
Constraints
                                deltaSpeed =
(qpg_VHC_speed(nearestAdjacentAheadVp) + relativeSpeed) - qpg_VHC_speed(Vp);
                                //if (vehicleDirectlyAhead != NULL) // check for
imminent collision with vehicle ahead (conflict between merging control logic and
car following logic)
                                //{
                                //      if (qpg_VHC_distance(Vp) -
qpg_VHC_distance(vehicleDirectlyAhead) < 4)
                                //      {
                                //              return CFM_Speed;
                                //      }
                                //}
                                if (deltaSpeed > max_accel_mpss*timeStep)
                                {
                                        return MAX((qpg_VHC_speed(Vp) +
max_accel_mpss*timeStep), 0);
                                }
                                else if (deltaSpeed < min_decel_mpss*timeStep)
                                {
                                        return MAX((qpg_VHC_speed(Vp) +
min_decel_mpss*timeStep), 0);
                                }
                                else
                                {
                                        return
MAX((qpg_VHC_speed(nearestAdjacentAheadVp) + relativeSpeed), 0); //uses speed of
vehicle in adjacent lane
                                }
```

```
                        }

                }
        }
    }
    else if (myVeh->merging == 1) // Stage 1 of merging
    {
        //set local variables
        nearestAdjacentBehindVp = myVeh->vehicleAdjacentBehindP;
        nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);
        nearestAdjacentAheadVp = myVeh->vehicleAdjacentAheadP;
//qpg_VHC_ahead(myVeh->vehicleAdjacentBehindP);
        nearestAdjacentAheadVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentAheadVp);


        if (vehicleDirectlyAhead != NULL) // check for imminent collision
with vehicle ahead (conflict between merging control logic and car following
logic)
        {
                //if the vehicle ahead is too close, or the vehicle ahead is
behind the bumper of the leader, then reset targetSpacing for current vehicle and
adjacent Behind vehicle
                if (((qpg_VHC_distance(Vp) -
qpg_VHC_distance(vehicleDirectlyAhead)) < 4) ||
((qpg_VHC_distance(vehicleDirectlyAhead) - (qpg_VHC_distance(myVeh-
>vehicleAdjacentAheadP))) > 0)) //took out vehicleLength...behavior should be
triggered as soon as the front bumper of the vehicleDirectlyAhead is behind the
front bumper of the leader
                {
                        myVeh->vehTargetSpacing = 2*targetSpacing +
vehicleLength; // same as incrementing by targetSpacing + vehicleLength
                        nearestAdjacentBehindVdata->vehTargetSpacing =
3*targetSpacing + 2*vehicleLength; // same as incrementing by targetSpacing +
vehicleLength
                }
        }

        //check if spacing has been achieved by nearestAdjacentBehindVdata
(check spacing between nearestAdjacentBehind and nearestAdjacentAhead)
        //currentClearance2 = qpg_VHC_distance(nearestAdjacentBehindVp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is on the lane adjacent to the
current vehicle
        //currentSpacing2 = currentClearance2 - vehicleLength;
        //adjacentTargetDistance = currentSpacing2 -
nearestAdjacentBehindVdata->vehTargetSpacing; // changed from targetSpacing

        //check spacing between nearestAdjacentBehind and current vehicle
        currentClearance2 = qpg_VHC_distance(nearestAdjacentBehindVp) -
qpg_VHC_distance(Vp); //first vehicle is on the lane adjacent to the current
vehicle
        currentSpacing2 = currentClearance2 - vehicleLength;
```

```
		adjacentTargetDistanceBehind = currentSpacing2 - targetSpacing;
//myVeh->vehTargetSpacing; // vehTargetSpacing should equal default (5)


		//Apply altered follower code to the current vehicle
		if (link == qpg_VHC_link(nearestAdjacentAheadVp))
		{
			currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is actually cross-lane clearance,
check for inaccuracies in this line
		}
		else //nearestAdjacentAhead vehicle has changed links
		{
			//Method 2 of calculating vehicle clearance when vehicles are
on consecutive links
			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1, &b1,
&g1);
			qpg_POS_vehicle(nearestAdjacentAheadVp,
qpg_VHC_link(nearestAdjacentAheadVp), &x2, &y2, &z2, &b2, &g2);
			currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1)); //
assumes both vehicles are the same type
		}
		currentSpacing = currentClearance - vehicleLength;
		targetDistance = currentSpacing - myVeh->vehTargetSpacing; //
changed from targetSpacing
		if (targetDistance > 0)
		{
			relativeSpeed = targetDistance/2;
		}
		else
		{
			relativeSpeed = targetDistance/2; //0.5 , use 2 for more
gradual deceleration
		}

		if (linkTransition == 1)//check if next link was reached before
manuever could be completed
		{
			qps_VHC_laneRange(Vp, 1, 1); //prevent vehicle from
immediately passing through middle lane

			qps_GUI_printf("Vehicle %d could not complete manuever,
merge_state = %d ", qpg_VHC_uniqueID(Vp), myVeh->merging);

			//Lane Change is complete, reset necessary parameters and
variables (this is the same code as in merging == 2, currentLane == 2)
			myVeh->merging = 0;
			nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(myVeh->vehicleAdjacentBehindP);
			//nearestAdjacentBehindVdata->vehTargetSpacing =
targetSpacing; // reset target spacing of vehicle to default value such that the
vehicle will now follow the newly merged vehicle
			nearestAdjacentBehindVdata->vehTargetSpacing =
MAX(targetSpacing, nearestAdjacentBehindVdata->vehTargetSpacing - (targetSpacing +
```

```
vehicleLength)); //decrement vehTargetSpacing to account for a single vehicle
merging (two vehicles may be attempting to merge at once)
                    myVeh->vehTargetSpacing = targetSpacing; // NOTE: check if
this causes crashes
                    //the currentClearance is used in the followers return value
section
                    if (link == qpg_VHC_link(myVeh->vehicleAhead)) //if current
vehicle and vehicle directly ahead are on the same link
                    {
                           currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(myVeh->vehicleAhead);
                    }
                    else //nearestAdjacentAhead vehicle has changed links
                    {
                           //Method 2 of calculating vehicle clearance when
vehicles are on consecutive links
                           qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1,
&b1, &g1);
                           qpg_POS_vehicle(myVeh->vehicleAhead,
qpg_VHC_link(myVeh->vehicleAhead), &x2, &y2, &z2, &b2, &g2);
                           currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-
y1)); // assumes both vehicles are the same type
                    }
             }
             else //no link transition, continue with relative longitudinal
positioning phase of merging
             {
                    //if ((abs(targetDistance) < 1) && (adjacentTargetDistance > -
1)) //if current vehicle is close to the desired control point, and the platoon
has finished splitting, then proceed with merging operation
                    if ((targetDistance > -1) && (adjacentTargetDistanceBehind > -
1) && ((qpg_VHC_speed(Vp) - qpg_VHC_speed(nearestAdjacentBehindVp)) > -2.2352)) //
if the current vehicle is sufficiently behind the adjacent ahead vehicle, and if
the current vehicle is sufficiently ahead of the adjacent behind vehicle, and the
current vehicle speed is within 5 mph of the adjacent behind vehicle speed
                    {
                           //qps_GUI_printf("entered 1_5");
                           //maybe set let in flag of nearestAdjacentBehind?

                           //Method 1: do gradual lane change using laneRanges
                           qps_VHC_laneRange(Vp, 2, 2); //check to see if this
works
                           myVeh->merging = 2;
                    }
                    else
                    {
                           //qps_GUI_printf("entered 1_6");
                           qps_VHC_laneRange(Vp, 1, 1); //prevent vehicle from
making premature lane change
                    }
                    //qps_GUI_simRunning(0); // works! 0 = false

                    //Apply Acceleration and Deceleration Constraints
```

```
                    deltaSpeed = (qpg_VHC_speed(nearestAdjacentAheadVp) +
relativeSpeed) - qpg_VHC_speed(Vp);
                    //if (vehicleDirectlyAhead != NULL) // check for imminent
collision with vehicle ahead (conflict between merging control logic and car
following logic)
                    //{
                    //     if (qpg_VHC_distance(Vp) -
qpg_VHC_distance(vehicleDirectlyAhead) < 4)
                    //     {
                    //            return CFM_Speed;
                    //     }
                    //}
                    if (deltaSpeed > max_accel_mpss*timeStep)
                    {
                           return MAX((qpg_VHC_speed(Vp) +
max_accel_mpss*timeStep), 0);
                    }
                    else if (deltaSpeed < min_decel_mpss*timeStep)
                    {
                           return MAX((qpg_VHC_speed(Vp) +
min_decel_mpss*timeStep), 0);
                    }
                    else
                    {
                           return MAX((qpg_VHC_speed(nearestAdjacentAheadVp) +
relativeSpeed), 0); //uses speed of vehicle in adjacent lane
                    }
             }
       }
       else if (myVeh->merging == 2) // check if merging is complete , change this
to target lane later
       {
             if (currentLane == 1) // merging in progress
             {
                    //qps_GUI_printf("entered 2");
                    //set local variables
                    //nearestAdjacentBehindVp = myVeh->vehicleAdjacentBehindP;
                    //nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentBehindVp);
                    nearestAdjacentAheadVp = myVeh->vehicleAdjacentAheadP;
//qpg_VHC_ahead(myVeh->vehicleAdjacentBehindP);
                    //nearestAdjacentAheadVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(nearestAdjacentAheadVp);

                    //Apply altered follower code to the current vehicle
                    if (link == qpg_VHC_link(nearestAdjacentAheadVp))
                    {
                           currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(nearestAdjacentAheadVp); //this is actually cross-lane clearance,
check for inaccuracies in this line
                    }
                    else //nearestAdjacentAhead vehicle has changed links
                    {
```

```
                            //Method 2 of calculating vehicle clearance when
vehicles are on consecutive links
                            qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1,
&b1, &g1);
                            qpg_POS_vehicle(nearestAdjacentAheadVp,
qpg_VHC_link(nearestAdjacentAheadVp), &x2, &y2, &z2, &b2, &g2);
                            currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-
y1)); // assumes both vehicles are the same type
                     }
                     currentSpacing = currentClearance - vehicleLength;
                     targetDistance = currentSpacing - myVeh->vehTargetSpacing; //
changed from targetSpacing
                     if (targetDistance > 0)
                     {
                            relativeSpeed = targetDistance/2;
                     }
                     else
                     {
                            relativeSpeed = targetDistance/2; //0.5 , use 2 for
more gradual deceleration
                     }


                     if (linkTransition == 1)//check if next link was reached
before manuever could be completed
                     {
                            qps_VHC_laneRange(Vp, 1, 1); //prevent vehicle from
immediately passing through middle lane

                            qps_GUI_printf("Vehicle %d could not complete manuever,
merge_state = %d ", qpg_VHC_uniqueID(Vp), myVeh->merging);

                            //Lane Change is complete, reset necessary parameters
and variables (this is the same code as in merging == 2, currentLane == 2)
                            myVeh->merging = 0;
                            nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(myVeh->vehicleAdjacentBehindP);
                            //nearestAdjacentBehindVdata->vehTargetSpacing =
targetSpacing; // reset target spacing of vehicle to default value such that the
vehicle will now follow the newly merged vehicle
                            nearestAdjacentBehindVdata->vehTargetSpacing =
MAX(targetSpacing, nearestAdjacentBehindVdata->vehTargetSpacing - (targetSpacing +
vehicleLength)); //decrement vehTargetSpacing to account for a single vehicle
merging (two vehicles may be attempting to merge at once)
                            myVeh->vehTargetSpacing = targetSpacing; // NOTE: check
if this causes crashes
                            //the currentClearance is used in the followers return
value section
                            if (link == qpg_VHC_link(myVeh->vehicleAhead)) //if
current vehicle and vehicle directly ahead are on the same link
                            {
                                   currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(myVeh->vehicleAhead);
                            }
```

```
                            else //nearestAdjacentAhead vehicle has changed links
                            {
                                    //Method 2 of calculating vehicle clearance when
vehicles are on consecutive links
                                    qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1,
&z1, &b1, &g1);
                                    qpg_POS_vehicle(myVeh->vehicleAhead,
qpg_VHC_link(myVeh->vehicleAhead), &x2, &y2, &z2, &b2, &g2);
                                    currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-
y1)*(y2-y1)); // assumes both vehicles are the same type
                            }
                    }
                    else //no link transition, continue with lateral transition
phase of merging
                    {
                            qps_VHC_laneRange(Vp, 2, 2); //check to see if this
works, this doesn't seem to always work, try additional methods
                            qps_VHC_laneChange(Vp, 1); // should set Want Left
flag, ...hopefully sets Let In flag for adjacent behind vehicle ...didn't work
                            qps_VHC_changeLane(Vp, 1); // try immediate lane change

                            //Apply Acceleration and Deceleration Constraints
                            deltaSpeed = (qpg_VHC_speed(nearestAdjacentAheadVp) +
relativeSpeed) - qpg_VHC_speed(Vp);
                            //if (vehicleDirectlyAhead != NULL) // check for
imminent collision with vehicle ahead (conflict between merging control logic and
car following logic)
                            //{
                            //      if (qpg_VHC_distance(Vp) -
qpg_VHC_distance(vehicleDirectlyAhead) < 4)
                            //      {
                            //              return CFM_Speed;
                            //      }
                            //}
                            if (deltaSpeed > max_accel_mpss*timeStep)
                            {
                                    return MAX((qpg_VHC_speed(Vp) +
max_accel_mpss*timeStep), 0);
                            }
                            else if (deltaSpeed < min_decel_mpss*timeStep)
                            {
                                    return MAX((qpg_VHC_speed(Vp) +
min_decel_mpss*timeStep), 0);
                            }
                            else
                            {
                                    return
MAX((qpg_VHC_speed(nearestAdjacentAheadVp) + relativeSpeed), 0); //uses speed of
vehicle in adjacent lane
                            }
                    }
            }
            else if (currentLane == 2) // merging complete (mostly)
            {
```

```
                    //qps_GUI_printf("entered 3");
                    //qps_GUI_simRunning(0); // works! 0 = false
                    //Lane Change is complete, reset necessary parameters and
variables
                    myVeh->merging = 0;
                    nearestAdjacentBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(myVeh->vehicleAdjacentBehindP);
                    //nearestAdjacentBehindVdata->vehTargetSpacing =
targetSpacing; // reset target spacing of vehicle to default value such that the
vehicle will now follow the newly merged vehicle
                    nearestAdjacentBehindVdata->vehTargetSpacing =
MAX(targetSpacing, nearestAdjacentBehindVdata->vehTargetSpacing - (targetSpacing +
vehicleLength)); //decrement vehTargetSpacing to account for a single vehicle
merging (two vehicles may be attempting to merge at once)
                    myVeh->vehTargetSpacing = targetSpacing; // NOTE: check if
this causes crashes
                    //the currentClearance is used in the followers return value
section
                    if (link == qpg_VHC_link(myVeh->vehicleAhead)) //if current
vehicle and vehicle directly ahead are on the same link
                    {
                           currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(myVeh->vehicleAhead);
                    }
                    else //nearestAdjacentAhead vehicle has changed links
                    {
                           //Method 2 of calculating vehicle clearance when
vehicles are on consecutive links
                           qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1,
&b1, &g1);
                           qpg_POS_vehicle(myVeh->vehicleAhead,
qpg_VHC_link(myVeh->vehicleAhead), &x2, &y2, &z2, &b2, &g2);
                           currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-
y1)); // assumes both vehicles are the same type
                    }
                    //qps_GUI_printf("Current Vehicle ID = %d ",
qpg_VHC_uniqueID(Vp));
                    //qps_GUI_printf("Adjacent Behind ID = %d ",
qpg_VHC_uniqueID(myVeh->vehicleAdjacentBehindP));
                    ////qps_GUI_printf("Adjacent Ahead ID = %d ",
qpg_VHC_uniqueID(nearestAdjacentAheadVp));
                    //qps_GUI_printf("Behind ID = %d ",
qpg_VHC_uniqueID(qpg_VHC_behind(Vp)));
                    //qps_GUI_printf("Ahead ID = %d ",
qpg_VHC_uniqueID(qpg_VHC_ahead(Vp)));
             }
       }
       else if (myVeh->vehicleAhead != NULL && currZone != 1 && currZone != 2 &&
(qpg_LNK_zone(qpg_VHC_link(myVeh->vehicleAhead)) != 11) && (currentLane != 1 ||
myVeh->laneDropApproaching == 0)) //last condition prohibits CACC platoon
formation on bottom lane of links with an impending lane reduction of one lane
...add condition to prevent platoon formation near entry ramps
       {
```

```
		//Calculate currentClearance (front bumper to front bumper distance)
		if (link == qpg_VHC_link(myVeh->vehicleAhead)) //if current vehicle
and vehicle directly ahead are on the same link
			currentClearance = qpg_VHC_distance(Vp) -
qpg_VHC_distance(ahead); //Note: clearance is distance between front bumpers of
two vehicles
		else if (qpg_LNK_nodeEnd(link) ==
qpg_LNK_nodeStart(qpg_VHC_link(myVeh->vehicleAhead))) //else if the vehicle
directly ahead is on the next link
		{
			//Method 1, Paramics bug!: end of current link not at same 2D
point as beginning of next link
			//currentClearance = qpg_VHC_distance(Vp) +
qpg_LNK_length(qpg_VHC_link(myVeh->vehicleAhead)) - qpg_VHC_distance(myVeh-
>vehicleAhead); //units are meters // this line fails, there is some space in
between links (~1.5 m)
			//Method 2 of calculating vehicle clearance when vehicles are
on consecutive links
			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1, &b1,
&g1);
			qpg_POS_vehicle(myVeh->vehicleAhead, qpg_VHC_link(myVeh-
>vehicleAhead), &x2, &y2, &z2, &b2, &g2);
			currentClearance = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1)); //
assumes both vehicles are the same type
		}
		else
		{
			currentClearance = 100; // some distance longer than the
threshold of following distance 20m (this case occurs if the vehicle ahead is more
than one link ahead, assuming all links are >= the triggering distance)
		}

		//Update CACC status accordingly
		if (myVeh->EQUIPPED_FLAG == 1 && myVeh->dest != 11) // check if
vehicle needs to leave platoon to reach desired exit, 2nd condition is to check if
zone is a mainstream zone (think about how to generalize this)
		{
			//... && currZone != myVeh->dest
			//qpg_ZNE_link(qpg_NET_zone(myVeh->dest), myVeh->dest);
			//qpg_ZNE_link(qpg_NET_zone(myVeh->dest),
qpg_ZNE_index(qpg_NET_zone(myVeh->dest))); // try this one ...still null
			//qpg_ZNE_index(ZONE* zone);

			//qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1); // try this one,
might work, assuming there is only one link associated with a particular zone

			//qpg_ZNE_links(qpg_NET_zone(myVeh->dest)); //test, hopefully
just one

			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1, &b1,
&g1);

	//qpg_POS_node(qpg_LNK_nodeStart(qpg_ZNE_link(qpg_NET_zone(myVeh->dest),
```

```
1)), &x5, &y5, &z5); //can move this to initialization code later, as a VEHICLE
field
                    //distToDestNode = sqrt((x5-x1)*(x5-x1) + (y5-y1)*(y5-y1));
//rough approximation of distance to destination zone
                    distToDestNode = sqrt((myVeh->x_destNode-x1)*(myVeh-
>x_destNode-x1) + (myVeh->y_destNode-y1)*(myVeh->y_destNode-y1)); //rough
approximation of distance to destination zone
                    //qps_GUI_printf("Vehicle %d trip %d -> %d on link %s;
destination link %s, destination node %s, destination node position %f , %f ; Case
= %s ; #of links associated with destination zone = %d\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)),
qpg_LNK_name(qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1)),
qpg_NDE_name(qpg_LNK_nodeStart(qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1))), x5,
y5, "Node Test", qpg_ZNE_links(qpg_NET_zone(myVeh->dest)));

                    //if (qpg_LNK_exit(link, qpg_VHC_nextExit(Vp)) != NULL)
                    //{
                    //     if ((qpg_LNK_category(qpg_LNK_exit(link,
qpg_VHC_nextExit(Vp))) == 70) && currZone != 4 && currZone != 6) // check if the
next link is a 2 lane exit, NOTE: change this to add any additional exit ramp
categories for other networks
                    //     {
                    if (distToDestNode < 1000 && currZone != myVeh->dest)
                    {

                                   myVeh->EXIT_APPROACHING_FLAG = 1; //used to
prevent the participation flag from being reset to 1 after becoming 0
                                   ////Leader/None-state correction
                                   //if (myVeh->CACCstatus == 1 && BehindVdata-
>CACCstatus != -1)
                                   //{
                                   //     myVeh->CACCstatus = 0;
                                   //}

                                   // if current vehicle is not in a platoon,
immediately switch the Participation flag
                                   if (myVeh->CACCstatus == 0)
                                   {
                                          myVeh->PARTICIPATION_FLAG = 0;
                                   }

                                   if (myVeh->CACCstatus == 1 && myVeh-
>vehicleBehindP != NULL && BehindVdata != NULL) //leader logic for leaving
platoon, make sure that the vehicle behind exists
                                   {
                                          if (myVeh->splitting == 0)
                                          {
                                                 //qps_GUI_printf("Vehicle %d trip
%d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "1-1");
                                                 //qps_GUI_simRunning(0);

                                                 myVeh->splitting = 1; //begin
platoon splitting manuever
```

```
                                                assert(myVeh->vehicleBehindP !=
NULL);
                                                /*if (BehindVdata == NULL)
                                                {
                                                        qps_GUI_printf("ERROR:
BehindVdata == NULL");
                                                        qps_GUI_simRunning(0);
                                                        assert(BehindVdata !=
NULL);
                                                }*/
                                                //have the vehicle behind the
leader increase his following distance before being given the platoon leader role
                                                BehindVdata->vehTargetSpacing =
2*targetSpacing + vehicleLength; //watch out for vehicle cutting into this gap
                                        }
                                        else if (myVeh->splitting == 1)
//assumption that current vehicle and vehicle behind are on the same link should
be valid (necessarily)
                                        {
                                                /*if (myVeh->vehicleBehindP ==
NULL)
                                                {
                                                        qps_GUI_printf("ERROR:
vehicleBehindP == NULL; Vehicle %d trip %d -> %d on link %s; Case = %s\n",
qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp), qpg_VHC_destination(Vp),
qpg_LNK_name(qpg_VHC_link(Vp)), "1-1b");
                                                        qps_GUI_simRunning(0);
                                                        assert(myVeh-
>vehicleBehindP != NULL);
                                                }*/

                                                //calculate currentRearClearance,
distance from front bumper of rear vehicle to front bumper of current vehicle
                                                if (link == qpg_VHC_link(myVeh-
>vehicleBehindP)) //if current vehicle and vehicle behind are on the same link
                                                {
                                                        currentRearClearance =
qpg_VHC_distance(myVeh->vehicleBehindP) - qpg_VHC_distance(Vp); //Note: clearance
is distance between front bumpers of two vehicles
                                                }
                                                else //vehicles are on different
links
                                                {
                                                        qpg_POS_vehicle(Vp,
qpg_VHC_link(Vp), &x3, &y3, &z3, &b3, &g3); //current vehicle
                                                        qpg_POS_vehicle(myVeh-
>vehicleBehindP, qpg_VHC_link(myVeh->vehicleBehindP), &x4, &y4, &z4, &b4, &g4);
//vehicle behind the current vehicle
                                                        currentRearClearance =
sqrt((x4-x3)*(x4-x3) + (y4-y3)*(y4-y3)); // assumes both vehicles are the same
type
                                                }
```

```
                                                if ((currentRearClearance -
vehicleLength) > (BehindVdata->vehTargetSpacing - 1))
                                                {
                                                        /*if (qpg_VHC_uniqueID(Vp)
== 67)
                                                        {

        qps_GUI_printf("Vehicle %d trip %d -> %d on link %s; Case = %s\n",
qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp), qpg_VHC_destination(Vp),
qpg_LNK_name(qpg_VHC_link(Vp)), "1-2");

        qps_GUI_simRunning(0);
                                                        }*/

                                                        myVeh->splitting = 2;
//manuever complete, update CACC roles accordingly
                                                        myVeh->CACCstatus = 0;
//current vehicle now is controlled by default Paramics logic on the way to its
exit
                                                        myVeh->PARTICIPATION_FLAG =
0; //prevent this vehicle from rejoining a platoon on the way to its exit

                                                        if (BehindVdata == NULL)
                                                        {

        qps_GUI_printf("ERROR: BehindVdata == NULL");

        qps_GUI_simRunning(0);
                                                                assert(BehindVdata
!= NULL);
                                                        }

                                                        BehindVdata-
>vehTargetSpacing = targetSpacing; //reset target spacing of vehicle behind
(necessary for case when vehicle behind may (re)join a platoon ahead)

                                                        //if (vehicleBehindBehind
!= NULL)
                                                        //{
                                                        //      BehindBehindVdata =
(VEHICLE_DATA*)qpg_VHC_userdata(vehicleBehindBehind);
                                                        //}
                                                        //if ( )//(myVeh-
>vehicleBehindP == qpg_LNK_vehicleTail(link, currentLane)) //if vehicle behind is
the tail of the platoon, set the vehicle behind CACCstatus to 0
                                                        //{
                                                        //      BehindVdata-
>CACCstatus = 0;
                                                        //}
                                                        //else
                                                        //{
                                                        BehindVdata->CACCstatus =
1; //vehicle behind current vehicle now becomes the platoon leader      ...what
about platoon of length 2?...if same exit, they should become a None State vehicle
```

```
                                                        //}
                                                }
                                        }
                                }
                                else if (myVeh->CACCstatus == -1) //follower
logic for leaving platoon
                                {
                                        //Cases
                                        //Case 1: current vehicle is the tail of
the platoon
                                        //Case 2: current vehicle is somewhere
else in the platoon

                                        if (BehindVdata == NULL || BehindVdata-
>CACCstatus != -1) // if the vehicle is the tail of the platoon (Case 1) ...NOTE:
BehindVdata may be null
                                        {
                                                if (myVeh->splitting == 0)
                                                {
                                                        //qps_GUI_printf("Vehicle
%d trip %d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "2-
1");
                                                        //qps_GUI_simRunning(0);

                                                        myVeh->splitting = 1;
//begin platoon splitting manuever
                                                        //have the current vehicle
increase his following distance before leaving the platoon
                                                        myVeh->vehTargetSpacing =
2*targetSpacing + vehicleLength; //watch out for vehicle cutting into this gap
                                                        myVeh->PARTICIPATION_FLAG =
0; //prevent this vehicle from rejoining a platoon on the way to its exit, also
prevent vehicle behind current vehicle from joining platoon (NOTE: this logic is
not exactly correct, since technically, the current vehicle participates in the
platoon until merging stage 2) ...this line may be unnecessary due to the
splitting variable already being set to 1
                                                }
                                                else if (myVeh->splitting == 1 &&
((currentClearance - vehicleLength) > (myVeh->vehTargetSpacing - 1))) //replaced
(qpg_VHC_distance(Vp) - qpg_VHC_distance(myVeh->vehicleAhead)) with
currentClearance (this incorporates the case of the two vehicles being on
subsequent separate links
                                                {
                                                        //qps_GUI_printf("Vehicle
%d trip %d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "2-
2");
                                                        //qps_GUI_simRunning(0);

                                                        myVeh->splitting = 2;
//manuever complete, update CACC roles accordingly
```

```
                                                        myVeh->CACCstatus = 0;
//current vehicle now is controlled by default Paramics logic on the way to its
exit

                                                        assert(vehAhead != NULL);
                                                        if (vehAhead->CACCstatus ==
1) //leader transitions to No role if the vehicle behind the leader splits the
platoon, this subcase may only occur if the original platoon is two vehicles long
                                                        {
                                                                vehAhead->CACCstatus
= 0; //may need to change the participation flag of the leader too
                                                        }

                                                        myVeh->vehTargetSpacing =
targetSpacing; // not actually necessary since vehicle never rejoins a platoon,
included for the sake of completeness
                                                }
                                        }
                                        else //BehindVdata->CACCstatus == -1 //
the vehicle is neither the leader, nor the tail, of the platoon (Case 2)
                                        {
                                                assert(myVeh->vehicleBehindP !=
NULL);
                                                assert(BehindVdata != NULL);
                                                if (myVeh->splitting == 0)
                                                {
                                                        //qps_GUI_printf("Vehicle
%d trip %d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "3-
1");
                                                        //qps_GUI_simRunning(0);

                                                        myVeh->splitting = 1;
//begin platoon splitting manuever
                                                        //have the vehicle behind
the current vehicle increase his following distance before the current vehicle
leaves the platoon
                                                        BehindVdata-
>vehTargetSpacing = 2*targetSpacing + vehicleLength; //watch out for vehicle
cutting into this gap
                                                        //have the current vehicle
increase his following distance before leaving the platoon
                                                        myVeh->vehTargetSpacing =
2*targetSpacing + vehicleLength;
                                                }
                                                else if (myVeh->splitting == 1 &&
((currentClearance - vehicleLength) > (myVeh->vehTargetSpacing - 1)) )
//replaced (qpg_VHC_distance(Vp) - qpg_VHC_distance(myVeh->vehicleAhead)) with
currentClearance (this incorporates the case of the two vehicles being on
subsequent separate links , moved rear vehicle check as further condition inside
this section
                                                {
```

```
                                                        //calculate
currentRearClearance, distance from front bumper of rear vehicle to front bumper
of current vehicle
                                                        if (link ==
qpg_VHC_link(myVeh->vehicleBehindP)) //if current vehicle and vehicle behind are
on the same link
                                                        {
                                                                currentRearClearance
= qpg_VHC_distance(myVeh->vehicleBehindP) - qpg_VHC_distance(Vp); //Note:
clearance is distance between front bumpers of two vehicles
                                                        }
                                                        else //vehicles are on
different links
                                                        {
                                                                qpg_POS_vehicle(Vp,
qpg_VHC_link(Vp), &x3, &y3, &z3, &b3, &g3); //current vehicle

        qpg_POS_vehicle(myVeh->vehicleBehindP, qpg_VHC_link(myVeh->vehicleBehindP),
&x4, &y4, &z4, &b4, &g4); //vehicle behind the current vehicle
                                                                currentRearClearance
= sqrt((x4-x3)*(x4-x3) + (y4-y3)*(y4-y3)); // assumes both vehicles are the same
type
                                                        }

                                                        if ((currentRearClearance -
vehicleLength) > (BehindVdata->vehTargetSpacing - 1))
                                                        {

        //qps_GUI_printf("Vehicle %d trip %d -> %d on link %s; Case = %s\n",
qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp), qpg_VHC_destination(Vp),
qpg_LNK_name(qpg_VHC_link(Vp)), "3-2");

        //qps_GUI_simRunning(0);

                                                                myVeh->splitting =
2; //manuever complete, update CACC roles accordingly
                                                                myVeh->CACCstatus =
0; //current vehicle now is controlled by default Paramics logic on the way to its
exit
                                                                myVeh-
>PARTICIPATION_FLAG = 0; //prevent this vehicle from rejoining a platoon on the
way to its exit
                                                                BehindVdata-
>CACCstatus = 1; //vehicle behind current vehicle now becomes the platoon leader

                                                                assert(vehAhead !=
NULL);
                                                                if (vehAhead-
>CACCstatus == 1) //leader transitions to No role if the vehicle behind the leader
splits the platoon
                                                                {
                                                                        vehAhead-
>CACCstatus = 0;
                                                                }
```

```
                                                                BehindVdata-
>vehTargetSpacing = targetSpacing; //reset target spacing of vehicle behind
(necessary for case when vehicle behind may (re)join a platoon ahead)
                                                                myVeh-
>vehTargetSpacing = targetSpacing; // not actually necessary since vehicle never
rejoins a platoon, included for the sake of completeness
                                                        }
                                                }
                                        }

                                }

                        //}
                }

                //test section to compensate for leaders failing to change
lanes during diverge protocol
                if (myVeh->splitting == 2 && currentLane == 5 && myVeh->dest
== 3) //just test for zone 3 for now
                {
                        if (distToDestNode < 500) //first try to encourage lane
change
                        {
                                qps_VHC_laneRange(Vp, 5, 1); //this should
already be automatically set by Paramics
                                qps_VHC_laneChange(Vp, -1); // should set Want
Left flag, ...hopefully sets Let In flag for adjacent behind vehicle ...didn't
work
                                //if (qpg_VHC_uniqueID(Vp) == 67)
                                //{
                                //     qps_GUI_printf("Vehicle %d trip %d -> %d
on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "C-1");
                                //     //qps_GUI_simRunning(0);
                                //}
                        }
                        if (distToDestNode < 500) // force lane change
                        {
                                qps_VHC_changeLane(Vp, -1); // try immediate
lane change, could cause problems
                                //if (qpg_VHC_uniqueID(Vp) == 67)
                                //{
                                //     qps_GUI_printf("Vehicle %d trip %d -> %d
on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "C-2");
                                //     //qps_GUI_simRunning(0);
                                //}
                        }
                }
        }

        //8/20/2014 add condition of similar velocity!, or else unstable
behavior ensues!
```

```
		relativeSpeed = qpg_VHC_speed(Vp) - qpg_VHC_speed(myVeh-
>vehicleAhead); //current vehicle must be within 10 mph of vehicle ahead to follow
it
		if (currentClearance <= followThreshold && myVeh->EQUIPPED_FLAG == 1
&& myVeh->PARTICIPATION_FLAG == 1 && vehAhead->EQUIPPED_FLAG == 1 && vehAhead-
>PARTICIPATION_FLAG == 1 && myVeh->splitting == 0 && (relativeSpeed > -2.2352*2))
//both vehicles must be equipped and participating to form platoon, leader vehicle
cannot become a follower if the splitting flag is set (this is to accomodate
unequipped vehicles merging)
		{
			myVeh->CACCstatus = -1; // current vehicle is set to be a
follower
			myVeh->platoonLaneIndex = qpg_VHC_lane(Vp); //check if this
works while the follower is changing lanes to join the platoon
			vehAhead->platoonLaneIndex = qpg_VHC_lane(Vp); //make sure
that leader does not change lanes during platoon formation
			if (vehAhead->CACCstatus == 0) // if vehicle ahead was at the
default state (neither leading nor following)
			{
				vehAhead->CACCstatus = 1; // set the state of the
vehicle ahead to be a leader
			}
			/*if (qpg_VHC_uniqueID(Vp) == 384)
			{
				qps_GUI_printf("Vehicle %d trip %d -> %d on link %s;
Case = %s, lane = %d, vehicleDirectlyAheadID = %d\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)),
"State Error", qpg_VHC_lane(Vp), qpg_VHC_uniqueID(qpg_VHC_ahead(Vp)));
				qps_GUI_simRunning(0);
			}*/
		}
	}
	else if (myVeh->vehicleAhead == NULL) //there is no vehicle ahead of the
current vehicle on the same link or part of the same platoon
	{
		//check if platoon splitting behavior is necessary

		//Update CACC status accordingly
		if (myVeh->EQUIPPED_FLAG == 1 && myVeh->dest != 11) // check if
vehicle needs to leave platoon to reach desired exit, 2nd condition is to check if
zone is a mainstream zone (think about how to generalize this)
		{
			qpg_POS_vehicle(Vp, qpg_VHC_link(Vp), &x1, &y1, &z1, &b1,
&g1);
			distToDestNode = sqrt((myVeh->x_destNode-x1)*(myVeh-
>x_destNode-x1) + (myVeh->y_destNode-y1)*(myVeh->y_destNode-y1)); //rough
approximation of distance to destination zone

			//if (qpg_LNK_exit(link, qpg_VHC_nextExit(Vp)) != NULL)
			//{
			//	if ((qpg_LNK_category(qpg_LNK_exit(link,
qpg_VHC_nextExit(Vp))) == 70) && currZone != 4 && currZone != 6) // check if the
next link is a 2 lane exit, NOTE: change this to add any additional exit ramp
categories for other networks
```

```
                    //     {
                    if (distToDestNode < 1000 && currZone != myVeh->dest)
                    {
                                    myVeh->EXIT_APPROACHING_FLAG = 1; //used to
prevent the participation flag from being reset to 1 after becoming 0

                                    // if current vehicle is not in a platoon,
immediately switch the Participation flag
                                    if (myVeh->CACCstatus == 0)
                                    {
                                            myVeh->PARTICIPATION_FLAG = 0;
                                    }

                                    if (myVeh->CACCstatus == 1 && myVeh-
>vehicleBehindP != NULL && BehindVdata != NULL) //leader logic for leaving
platoon, make sure that the vehicle behind exists
                                    {
                                            if (myVeh->splitting == 0)
                                            {
                                                    //qps_GUI_printf("Vehicle %d trip
%d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp),
qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "4-1");
                                                    //qps_GUI_simRunning(0);

                                                    myVeh->splitting = 1; //begin
platoon splitting manuever
                                                    //have the vehicle behind the
leader increase his following distance before being given the platoon leader role
                                                    BehindVdata->vehTargetSpacing =
2*targetSpacing + vehicleLength; //watch out for vehicle cutting into this gap
                                            }
                                            else if (myVeh->splitting == 1)
//assumption that current vehicle and vehicle behind are on the same link should
be valid (necessarily)
                                            {
                                                    //calculate currentRearClearance,
distance from front bumper of rear vehicle to front bumper of current vehicle
                                                    if (link == qpg_VHC_link(myVeh-
>vehicleBehindP)) //if current vehicle and vehicle behind are on the same link
                                                    {
                                                            currentRearClearance =
qpg_VHC_distance(myVeh->vehicleBehindP) - qpg_VHC_distance(Vp); //Note: clearance
is distance between front bumpers of two vehicles
                                                    }
                                                    else //vehicles are on different
links
                                                    {
                                                            qpg_POS_vehicle(Vp,
qpg_VHC_link(Vp), &x3, &y3, &z3, &b3, &g3); //current vehicle
                                                            qpg_POS_vehicle(myVeh-
>vehicleBehindP, qpg_VHC_link(myVeh->vehicleBehindP), &x4, &y4, &z4, &b4, &g4);
//vehicle behind the current vehicle
```

```
                                                        currentRearClearance =
sqrt((x4-x3)*(x4-x3) + (y4-y3)*(y4-y3)); // assumes both vehicles are the same
type
                                                }

                                                if ((currentRearClearance -
vehicleLength) > (BehindVdata->vehTargetSpacing - 1))
                                                {
                                                        //qps_GUI_printf("Vehicle
%d trip %d -> %d on link %s; Case = %s\n", qpg_VHC_uniqueID(Vp),
qpg_VHC_origin(Vp), qpg_VHC_destination(Vp), qpg_LNK_name(qpg_VHC_link(Vp)), "4-
2");
                                                        //qps_GUI_simRunning(0);

                                                        myVeh->splitting = 2;
//manuever complete, update CACC roles accordingly
                                                        myVeh->CACCstatus = 0;
//current vehicle now is controlled by default Paramics logic on the way to its
exit
                                                        myVeh->PARTICIPATION_FLAG =
0; //prevent this vehicle from rejoining a platoon on the way to its exit
                                                        BehindVdata->CACCstatus =
1; //vehicle behind current vehicle now becomes the platoon leader
                                                        BehindVdata-
>vehTargetSpacing = targetSpacing; //reset target spacing of vehicle behind
(necessary for case when vehicle behind may (re)join a platoon ahead)
                                                }
                                        }
                                }

                        //}
                }
        }
    }

    ////Leader/None-state correction
    //if (myVeh->CACCstatus == 1 && BehindVdata->CACCstatus != -1)
    //{
    //     myVeh->CACCstatus = 0;
    //}

    //color vehicle state section
    //if (myVeh->PARTICIPATION_FLAG != 0)
    //{
    //     if (myVeh->CACCstatus == 1) // leader
    //     {
    //            qps_DRW_vehicleColour(Vp, 4280791591); //medium green
4280791591 (0xFF27B227) //forest green 4279857945 (0xFF197319) //light green
4278255360 (OxFF00FF00)
    //     }
    //     else if (myVeh->CACCstatus == -1) // follower
    //     {
    //            qps_DRW_vehicleColour(Vp, 4278255615); //cyan 4294967040
(0xFFFFFF00) //yellow 4278255615 (0xFF00FFFF) //blue 4294901760 (OxFFFF0000)
```

```
	//	}
	//}

	//acceleration color section
	//qps_GUI_printf("timeStep duration is: %f seconds", timeStep);
	acceleration = (qpg_VHC_speed(Vp) - myVeh->prevSpeed)/(float)timeStep;
	//if (currZone != 1 && abs(acceleration) >= 1)
	//{
	//	qps_GUI_printf("acceleration is: %f m/s^2", acceleration); //
acceleration m/s^2
	//}
	//colorScale = (MAX(MIN(acceleration,10),-10)+10)/20;
	//qps_DRW_vehicleColour(Vp, qpg_DRW_colourScale(colorScale));
//colourScale, red is 1, blue is 0, other colors are in between too

	myVeh->prevSpeed = qpg_VHC_speed(Vp);

	//speed color section
	//colorScale = (MAX(MIN(qpg_VHC_speed(Vp),32),17)-17)/15;
	//qps_DRW_vehicleColour(Vp, qpg_DRW_colourScale(colorScale));
//colourScale, red is 1, blue is 0, other colors are in between too

	//vehicle tag section
	//qps_DRW_vehicleTag(Vp, qpg_DRW_colourScale(colorScale), 1, 10, "tag");
//doesn't work

	// return value section
	if (currZone != 1 && currZone != 2) //if vehicle is NOT in a zone (vehicle
is on a regular link)
	{
		if (myVeh->CACCstatus == 1 || myVeh->CACCstatus == 0)
		{
			return CFM_Speed; //leader keeps default car following speed
(same with default state)
		}
		else if (myVeh->CACCstatus == -1) // follower kinematic update
		{
			if (myVeh->vehicleAhead != NULL &&
qpg_LNK_zone(qpg_VHC_link(myVeh->vehicleAhead)) != 11) // changed from ahead to
myVeh->vehicleAhead // second condition permits vehicles to leave the network
without stalling
			{
				//qps_GUI_printf("speed = %f ", qpg_VHC_speed(myVeh-
>vehicleAhead));
				if (currentClearance <= 0)
				{
					qps_GUI_printf("currentClearance is <= 0");
				}
				currentSpacing = currentClearance - vehicleLength;
				targetDistance = currentSpacing - myVeh-
>vehTargetSpacing; // changed from targetSpacing
				if (targetDistance > 0) relativeSpeed =
targetDistance/2;
```

```
                            else relativeSpeed = targetDistance/3; //0.5//perhaps
add flag for platoon splitting to prevent sudden decelerations to 0 speed due to
attempting to reach a control point behind the current vehicle's position
                            //return MAX(CFM_Speed, qpg_VHC_speed(ahead) +
relativeSpeed); // try to change this to take out the max, or at least find
condition where vehicle ahead speed is 0 (Haitao's line)
                            //Apply Acceleration and Deceleration Constraints
                            deltaSpeed = (qpg_VHC_speed(myVeh->vehicleAhead) +
relativeSpeed) - qpg_VHC_speed(Vp);
                            if (deltaSpeed > max_accel_mpss*timeStep)
                            {
                                    return MAX((qpg_VHC_speed(Vp) +
max_accel_mpss*timeStep), 0);
                            }
                            else if (deltaSpeed < min_decel_mpss*timeStep)
                            {
                                    return MAX((qpg_VHC_speed(Vp) +
min_decel_mpss*timeStep), 0);
                            }
                            else
                            {
                                    return MAX((qpg_VHC_speed(myVeh->vehicleAhead) +
relativeSpeed), 0); //pure approach code (no default car-following speed used)
..., changed from ahead to myVeh->vehicleAhead
                            }
                    }
                    else
                    {
                            /*if (qpg_LNK_zone(qpg_VHC_link(myVeh->vehicleAhead))
!= 2)
                            {
                                    qps_GUI_printf("ERROR: default speed returned");
                            }*/
                            return CFM_Speed; // safe return value, occurs when
vehicleAhead has left the network
                    }
            }
            else return CFM_Speed; // default return value...this should never
be reached
    }
    else return CFM_Speed; // if vehicle is in a zone

}

float qpo_CFM_leadSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
    float CFM_Speed = qpg_CFM_leadSpeed(link, Vp, ahead);
    return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
}

float qpo_CFM_followSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
    float CFM_Speed = qpg_CFM_followSpeed(link, Vp, ahead);
    return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
```

```
}

void qpx_VHC_arrive(VEHICLE* Vp, LINK* link, ZONE* zone)
{
	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp);
	TT_veh[counter_arrived] = qpg_VHC_existTime(Vp);
	VMT += myVeh->tripDist;
	VHT += TT_veh[counter_arrived];

	if (myVeh->EQUIPPED_FLAG == 1)
	{
		VMT_e += myVeh->tripDist;
		VHT_e += TT_veh[counter_arrived];
		counter_arrived_e ++;
	}
	else //EQUIPPED_FLAG == 0
	{
		VMT_n += myVeh->tripDist;
		VHT_n += TT_veh[counter_arrived];
		counter_arrived_n ++;
	}

	if (myVeh->PARTICIPATED_FLAG == 1)
	{
		VHT_d += TT_veh[counter_arrived];
	}
	else //PARTICIPATED_FLAG == 0
	{
		VHT_nd += TT_veh[counter_arrived];
	}

	if ((myVeh->origin == 1 || myVeh->origin == 21) && myVeh->dest == 11)
	{
		if (myVeh->PARTICIPATED_FLAG == 1 && (qpg_VHC_lane(Vp) == 4))
//second condition is optional (network specific too...this can be changed if
necessary) ...vehicle ends in dedicated lane (vehicle may temporarily change lanes
before returning to the dedicated lane, but this should not negatively impact
energy)
		{
			VHT_d_mainstream += TT_veh[counter_arrived];
			energy_sum_d_mainstream += myVeh->total_energy;
			dist_sum_d_mainstream += myVeh->total_dist;
			counter_arrived_d_mainstream++;
		}
		else //PARTICIPATED_FLAG == 0
		{
			VHT_nd_mainstream += TT_veh[counter_arrived];
			energy_sum_nd_mainstream += myVeh->total_energy;
			dist_sum_nd_mainstream += myVeh->total_dist;
			counter_arrived_nd_mainstream++;
		}
	}

	counter_arrived++;
```

```
	free(myVeh);
}

void	qpx_NET_complete()
{
	FILE *fsbs, *fsum, *fopMode, *fVSP, *fTT, *fsbs_e, *fsum_e, *fopMode_e,
*fVSP_e, *fTT_e, *fsbs_n, *fsum_n, *fopMode_n, *fVSP_n, *fTT_n;
	int i;

	VMT = VMT*meter2mile;
	dist_sum = dist_sum*meter2mile;
	energy_sum = energy_sum/dist_sum/3600;
	CO2_sum = CO2_sum/dist_sum/3600;
	CO_sum = CO_sum/dist_sum/3600;
	HC_sum = HC_sum/dist_sum/3600;
	NOx_sum = NOx_sum/dist_sum/3600;
	PM_sum = PM_sum/dist_sum/3600;

	if (counter_arrived_e > 0)
	{
		//update equipped vehicle statistics
		VMT_e = VMT_e*meter2mile;
		dist_sum_e = dist_sum_e*meter2mile;
		energy_sum_e = energy_sum_e/dist_sum_e/3600;
		CO2_sum_e = CO2_sum_e/dist_sum_e/3600;
		CO_sum_e = CO_sum_e/dist_sum_e/3600;
		HC_sum_e = HC_sum_e/dist_sum_e/3600;
		NOx_sum_e = NOx_sum_e/dist_sum_e/3600;
		PM_sum_e = PM_sum_e/dist_sum_e/3600;

		//update dedicate lane(s) statistics
		VMT_d = VMT_d*meter2mile;
		dist_sum_d = dist_sum_d*meter2mile;
		energy_sum_d = energy_sum_d/dist_sum_d/3600;
		CO2_sum_d = CO2_sum_d/dist_sum_d/3600;
		CO_sum_d = CO_sum_d/dist_sum_d/3600;
		HC_sum_d = HC_sum_d/dist_sum_d/3600;
		NOx_sum_d = NOx_sum_d/dist_sum_d/3600;
		PM_sum_d = PM_sum_d/dist_sum_d/3600;

		//update non-dedicated lane statistics
		VMT_nd = VMT_nd*meter2mile;
		dist_sum_nd = dist_sum_nd*meter2mile;
		energy_sum_nd = energy_sum_nd/dist_sum_nd/3600;
		CO2_sum_nd = CO2_sum_nd/dist_sum_nd/3600;
		CO_sum_nd = CO_sum_nd/dist_sum_nd/3600;
		HC_sum_nd = HC_sum_nd/dist_sum_nd/3600;
		NOx_sum_nd = NOx_sum_nd/dist_sum_nd/3600;
		PM_sum_nd = PM_sum_nd/dist_sum_nd/3600;


		//update dedicated lane vehicle statistics
		dist_sum_d_mainstream = dist_sum_d_mainstream*meter2mile;
```

```
		energy_sum_d_mainstream =
energy_sum_d_mainstream/dist_sum_d_mainstream/3600;

		//update non-dedicated lane vehicle statistics
		dist_sum_nd_mainstream = dist_sum_nd_mainstream*meter2mile;
		energy_sum_nd_mainstream =
energy_sum_nd_mainstream/dist_sum_nd_mainstream/3600;
	}

	if (counter_arrived_n > 0)
	{
		VMT_n = VMT_n*meter2mile;
		dist_sum_n = dist_sum_n*meter2mile;
		energy_sum_n = energy_sum_n/dist_sum_n/3600;
		CO2_sum_n = CO2_sum_n/dist_sum_n/3600;
		CO_sum_n = CO_sum_n/dist_sum_n/3600;
		HC_sum_n = HC_sum_n/dist_sum_n/3600;
		NOx_sum_n = NOx_sum_n/dist_sum_n/3600;
		PM_sum_n = PM_sum_n/dist_sum_n/3600;
	}

	/*qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum, CO2_sum,
CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived, VMT/counter_arrived);
	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_e, CO2_sum_e,
CO_sum_e, HC_sum_e, NOx_sum_e, PM_sum_e, VHT_e/counter_arrived_e,
VMT_e/counter_arrived_e);
	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_n, CO2_sum_n,
CO_sum_n, HC_sum_n, NOx_sum_n, PM_sum_n, VHT_n/counter_arrived_n,
VMT_n/counter_arrived_n);

	qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
	qps_GUI_printf("%d equipped vehicles released. %d equipped vehicles
arrived.\n", counter_released_e,counter_arrived_e);
	qps_GUI_printf("%d non-equipped vehicles released. %d non-equipped vehicles
arrived.\n", counter_released_n,counter_arrived_n);*/


	//fsbs = fopen(out1Path, "w");
	fsum = fopen(out2Path, "w");
	/*fopMode = fopen(out3Path, "w");
	fVSP = fopen(out4Path, "w");
	fTT = fopen(out5Path, "w");
	fsbs_e = fopen(out11Path, "w");*/
	fsum_e = fopen(out21Path, "w");
	/*fopMode_e = fopen(out31Path, "w");
	fVSP_e = fopen(out41Path, "w");
	fTT_e = fopen(out51Path, "w");
	fsbs_n = fopen(out12Path, "w");*/
	fsum_n = fopen(out22Path, "w");
	/*fopMode_n = fopen(out32Path, "w");
	fVSP_n = fopen(out42Path, "w");
	fTT_n = fopen(out52Path, "w");*/
```

```
	//if (fsum == NULL || fopMode == NULL || fVSP == NULL || fTT == NULL ||
fsbs == NULL)
	//	qps_GUI_printf("Couldn't open file. (simulation ended)");

	//if (fsum == NULL || fopMode == NULL || fVSP == NULL || fTT == NULL ||
fsbs == NULL || fsum_e == NULL || fopMode_e == NULL || fVSP_e == NULL || fTT_e ==
NULL || fsbs_e == NULL || fsum_n == NULL || fopMode_n == NULL || fVSP_n == NULL ||
fTT_n == NULL || fsbs_n == NULL)
	if (fsum == NULL || fsum_e == NULL || fsum_n == NULL)
	{
		qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);
		qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_e,
CO2_sum_e, CO_sum_e, HC_sum_e, NOx_sum_e, PM_sum_e, VHT_e/counter_arrived_e,
VMT_e/counter_arrived_e);
		qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_n,
CO2_sum_n, CO_sum_n, HC_sum_n, NOx_sum_n, PM_sum_n, VHT_n/counter_arrived_n,
VMT_n/counter_arrived_n);

		qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
		qps_GUI_printf("%d equipped vehicles released. %d equipped vehicles
arrived.\n", counter_released_e,counter_arrived_e);
		qps_GUI_printf("%d non-equipped vehicles released. %d non-equipped
vehicles arrived.\n", counter_released_n,counter_arrived_n);
		qps_GUI_printf("Couldn't open file. (simulation ended, results not
logged)");
	}

	fprintf(fsum,
"Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi),VHT(sec/veh),VMT(
mi/veh)\n");
	if (counter_arrived > 0)
	{
		fprintf(fsum, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);
		fprintf(fsum, "%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
	}

	fprintf(fsum_e,
"Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi),VHT(sec/veh),VMT(
mi/veh)\n");
	if (counter_arrived_e > 0)
	{
		fprintf(fsum_e, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_e,
CO2_sum_e, CO_sum_e, HC_sum_e, NOx_sum_e, PM_sum_e, VHT_e/counter_arrived_e,
VMT_e/counter_arrived_e);
		fprintf(fsum_e, "%d equipped vehicles released. %d equipped vehicles
arrived.\n", counter_released_e,counter_arrived_e);
```

```
		fprintf(fsum_e, "%f, %f, %f, %f, %f, %f, %f, %f (dedicated lane)\n",
energy_sum_d, CO2_sum_d, CO_sum_d, HC_sum_d, NOx_sum_d, PM_sum_d,
VHT_d/counter_participated, VMT_d/counter_participated);
		fprintf(fsum_e, "%f, %f, %f, %f, %f, %f, %f, %f (non-dedicated
lanes)\n", energy_sum_nd, CO2_sum_nd, CO_sum_nd, HC_sum_nd, NOx_sum_nd, PM_sum_nd,
VHT_nd/(counter_arrived_e - counter_participated), VMT_nd/(counter_arrived_e -
counter_participated));
		fprintf(fsum_e, "%d equipped vehicles in dedicated lane.\n",
counter_participated);
		fprintf(fsum_e, "%d equipped vehicles with route (1->2) used
dedicated lane with average TT of %f seconds & average energy of %f KJ/mi\n",
counter_arrived_d_mainstream, VHT_d_mainstream/counter_arrived_d_mainstream,
energy_sum_d_mainstream);
		fprintf(fsum_e, "%d equipped vehicles with route (1->2) did not use
dedicated lane with average TT of %f seconds & average energy of %f KJ/mi\n",
counter_arrived_nd_mainstream, VHT_nd_mainstream/counter_arrived_nd_mainstream,
energy_sum_nd_mainstream);
	}

	fprintf(fsum_n,
"Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi),VHT(sec/veh),VMT(
mi/veh)\n");
	if (counter_arrived_n > 0)
	{
		fprintf(fsum_n, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_n,
CO2_sum_n, CO_sum_n, HC_sum_n, NOx_sum_n, PM_sum_n, VHT_n/counter_arrived_n,
VMT_n/counter_arrived_n);
		fprintf(fsum_n, "%d non-equipped vehicles released. %d non-equipped
vehicles arrived.\n", counter_released_n,counter_arrived_n);
	}


	/*for (i = 0; i <= 40; i++)
		fprintf(fopMode, "%d, %d\n", i, modeBins[i]);

	for (i = 0; i < counter_sbs; i++)
		fprintf(fVSP, "%f\n", VSP_sbs[i]);

	for (i = 0; i < counter_arrived; i++)
		fprintf(fTT, "%f\n", TT_veh[i]);*/

	/*fprintf(fsbs, "timeStamp, vehID, vehType, linkID, originZone, destZone,
speed(m/s), acc(m/s^2)\n");
	for (i = 0; i < counter_sbs; i++)
		fprintf(fsbs, "%d, %d, %d, %d, %d, %d, %f, %f\n", timeStamp_sbs[i],
vehID_sbs[i], vehType_sbs[i], linkID_sbs[i], origin_sbs[i], dest_sbs[i],
speed_sbs[i], acc_sbs[i]);

	fprintf(fsbs, "vehID, vehType, speed(m/s),
Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi)\n");
	for (i = 0; i < counter_sbs; i++)
		if (origin_sbs[i] == 40 && dest_sbs[i] == 12) fprintf(fsbs, "%d, %d,
%f\n", vehID_sbs[i], vehType_sbs[i], speed_sbs[i]);
	*/
```

```
	//fclose(fsbs);
	fclose(fsum);
	/*fclose(fopMode);
	fclose(fVSP);
	fclose(fTT);*/

	//fclose(fsbs_e);
	fclose(fsum_e);
	/*fclose(fopMode_e);
	fclose(fVSP_e);
	fclose(fTT_e);*/

	//fclose(fsbs_n);
	fclose(fsum_n);
	/*fclose(fopMode_n);
	fclose(fVSP_n);
	fclose(fTT_n);*/

	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum, CO2_sum,
CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived, VMT/counter_arrived);
	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_e, CO2_sum_e,
CO_sum_e, HC_sum_e, NOx_sum_e, PM_sum_e, VHT_e/counter_arrived_e,
VMT_e/counter_arrived_e);
	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_n, CO2_sum_n,
CO_sum_n, HC_sum_n, NOx_sum_n, PM_sum_n, VHT_n/counter_arrived_n,
VMT_n/counter_arrived_n);

	qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
	qps_GUI_printf("%d equipped vehicles released. %d equipped vehicles
arrived.\n", counter_released_e,counter_arrived_e);
	qps_GUI_printf("%d non-equipped vehicles released. %d non-equipped vehicles
arrived.\n", counter_released_n,counter_arrived_n);

	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f (dedicated lane)\n",
energy_sum_d, CO2_sum_d, CO_sum_d, HC_sum_d, NOx_sum_d, PM_sum_d,
VHT_d/counter_participated, VMT_d/counter_participated); //NOTE: need to add
participated Flag to calculate VHT_d!
	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f (non-dedicated lanes)\n",
energy_sum_nd, CO2_sum_nd, CO_sum_nd, HC_sum_nd, NOx_sum_nd, PM_sum_nd,
VHT_nd/(counter_arrived_e - counter_participated), VMT_nd/(counter_arrived_e -
counter_participated));
	qps_GUI_printf("%d equipped vehicles in dedicated lane.\n",
counter_participated);

	qps_GUI_printf("%d equipped vehicles with route (1/21->11) used dedicated
lane with average TT of %f seconds & average energy of %f KJ/mi\n",
counter_arrived_d_mainstream, VHT_d_mainstream/counter_arrived_d_mainstream,
energy_sum_d_mainstream);
	qps_GUI_printf("%d equipped vehicles with route (1/21->11) did not use
dedicated lane with average TT of %f seconds & average energy of %f KJ/mi\n",
counter_arrived_nd_mainstream, VHT_nd_mainstream/counter_arrived_nd_mainstream,
energy_sum_nd_mainstream);
}
```

Appendix C: Isolated Intersection Connected Vehicle Signal Optimization Code

The following code is C/Paramics plugin code for isolated intersection connected vehicle signal optimization.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "programmer.h"
#include <windows.h>
#include <assert.h>

#define MIN(a,b)        ((a) < (b) ? (a) : (b))
#define MAX(a,b)        ((a) > (b) ? (a) : (b))
#define NDEBUG //comment out this line to activate assert statements

typedef struct veh_profile VEHICLE_DATA;
struct veh_profile
{
	float A; // rolling coefficient in kW*sec/meter
	float B; // rotation coefficient in kW*sec^2/meter^2
	float C; // drag coefficient in kW*sec^3/meter^3
	float M; // vehicle source mass in metric tons
	float f; // fixed mass factor in metric tons
	int sourceType; // vehicle source type
	int ID; // vehicle ID
	float theta; // road grade angle
	float vel[4]; // 3-second velocities
	float acc[4]; // 3-second accelerations
	int linkID[4]; // 3-second link IDs
	float VSP[4]; // 3-second VSPs
	int mode[4]; // 3-second modes
	float tm[4]; // 3-second time stamps
	int first2sec; // the first 2 seconds vehicle enters the network
	float energy[4], CO2[4], CO[4], HC[4], NOx[4], PM[4]; // 3-second emission
data
	int origin; // vehicle's origin zone
	int dest; // vehicle's destination zone
	float x_destNode; // x coordinate of start node of vehicle's destination
zone
	float y_destNode; // y coordinate of start node of vehicle's destination
zone
	float tripDist; // vehicle's total trip distance
	int bound; // 1: northbound; 2: southbound: 0 otherwise
	LINK* currentLink; //pointer to most recent current link
```

```
        float prevSpeed; //used to calculate acceleration
        float total_energy; //trip energy for ego-vehicle
        float total_dist; //trip distance for ego-vehicle (may replace tripDist)
};


//Parameters (for Adaptive/Optimal Signal Control)

//Adaptive/Optimal Signal Control Variables
NODE* n[1]; //change this number to total number of signals within the network

//declare pointers
LINK* WEST_in; //check if this works
LINK* NORTH_in;
LINK* EAST_in;
LINK* SOUTH_in;

LINK* WEST_out;
LINK* NORTH_out;
LINK* EAST_out;
LINK* SOUTH_out;

//Independent NEMA phases (2 through-lane version)
//NOTE: These strings differ from the SUMO strings, due to the fact that Paramics
does not have independent lane control like SUMO does
//      Therefore, the characters in the SUMO strings refer to lanes, whereas the
characters in the Paramics strings below refer to movements (total of 12
movements)
//      One beneficial byproduct is that the Paramics strings remain the same for
all 4-way intersections regardless of the number of lanes, or if certain lanes
share movements.
//Description: The strings use the following characters: 1) G = green, 2) y =
yellow, 3) R = Right Turn On Red, 4) r = red
//              The movements are ordered cw, starting from the North_in right turn
movement, or more explicitly the 12 movements are as follows (NEMA phase
equivalent in parantheses):
//                  1. North_in, West_out  (4)
//                  2. North_in, South_out (4)
//                  3. North_in, East_out  (7)
//                  4. East_in, North_out  (6)
//                  5. East_in, West_out   (6)
//                  6. East_in, South_out  (1)
//                  7. South_in, East_out  (8)
//                  8. South_in, North_out (8)
//                  9. South_in, West_out  (3)
//                 10. West_in, South_out  (2)
//                 11. West_in, East_out   (2)
//                 12. West_in, North_out  (5)
//// without RTOR logic:
//const char *ONEGREEN = "rrrrrGrrrrrr";//"rrrrrrrGrrrrrrrr";
//const char *ONEYELLOW = "rrrrryrrrrrr";//"rrrrrrryrrrrrrrr";
//const char *TWOGREEN = "rrrrrrrrrGGr";//"rrrrrrrrrrrrGGGr";
//const char *TWOYELLOW = "rrrrrrrrryyr";//"rrrrrrrrrrrrryyyr";
//const char *THREEGREEN = "rrrrrrrrGrrr";//"rrrrrrrrrrrGrrrr";
```

```
//const char *THREEYELLOW = "rrrrrrrryrrr";//"rrrrrrrrrrryrrrr";
//const char *FOURGREEN = "GGrrrrrrrrrr";//"GGGrrrrrrrrrrrrr";
//const char *FOURYELLOW = "yyrrrrrrrrrr";//"yyyrrrrrrrrrrrrr";
//const char *FIVEGREEN = "rrrrrrrrrrrG";//"rrrrrrrrrrrrrrrG";
//const char *FIVEYELLOW = "rrrrrrrrrrry";//"rrrrrrrrrrrrrrry";
//const char *SIXGREEN = "rrrGGrrrrrrr";//"rrrrGGGrrrrrrrrr";
//const char *SIXYELLOW = "rrryyrrrrrrr";//"rrrryyyrrrrrrrrr";
//const char *SEVENGREEN = "rrGrrrrrrrrr";//"rrrGrrrrrrrrrrrr";
//const char *SEVENYELLOW = "rryrrrrrrrrr";//"rrryrrrrrrrrrrrr";
//const char *EIGHTGREEN = "rrrrrrGGrrrr";//"rrrrrrrrGGGrrrrr";
//const char *EIGHTYELLOW = "rrrrrrryyrrr";//"rrrrrrrryyyrrrrr";
//
//const char *ALLRED = "rrrrrrrrrrrr";//"rrrrrrrrrrrrrrrr";

// with RTOR logic: combination rule: G > y > r > R
const char *ONEGREEN = "RrrRrGRrrrrr";//"rrrrrrrGrrrrrrrr";
const char *ONEYELLOW = "RrrRryRrrrrr";//"rrrrrrryrrrrrrrr";
const char *TWOGREEN = "RrrRrrRrrGGr";//"rrrrrrrrrrrrGGGr";
const char *TWOYELLOW = "RrrRrrRrryyr";//"rrrrrrrrrrrryyyr";
const char *THREEGREEN = "rrrRrrRrGRrr";//"rrrrrrrrrrrGrrrr";
const char *THREEYELLOW = "rrrRrrRryRrr";//"rrrrrrrrrrryrrrr";
const char *FOURGREEN = "GGrRrrRrrRrr";//"GGGrrrrrrrrrrrrr";
const char *FOURYELLOW = "yyrRrrRrrRrr";//"yyyrrrrrrrrrrrrr";
const char *FIVEGREEN = "RrrrrrRrrRrG";//"rrrrrrrrrrrrrrrG";
const char *FIVEYELLOW = "RrrrrrRrrRry";//"rrrrrrrrrrrrrrry";
const char *SIXGREEN = "RrrGGrRrrRrr";//"rrrrGGGrrrrrrrrr";
const char *SIXYELLOW = "RrryyrRrrRrr";//"rrrryyyrrrrrrrrr";
const char *SEVENGREEN = "RrGRrrrrrRrr";//"rrrGrrrrrrrrrrrr";
const char *SEVENYELLOW = "RryRrrrrrRrr";//"rrryrrrrrrrrrrrr";
const char *EIGHTGREEN = "RrrRrrGGrRrr";//"rrrrrrrrGGGrrrrr";
const char *EIGHTYELLOW = "RrrRrryyrRrr";//"rrrrrrrryyyrrrrr";

const char *ALLRED = "rrrrrrrrrrrr";//"rrrrrrrrrrrrrrrr";

//Constants
const float GRAV = 9.8; // gravity coefficient in meter/second^2
const float INDEX = 2.23693629; // conversion index from m/s to mph
const float mps2mph = 2.23693629; //same as INDEX (consolidate this later)
const float meter2mile = 0.000621371;
const float vehicleLength = 3.9990; //4.8768 was for SUMO; // can replact this by
using the qpg_VHC_length(VEHICLE* vehicle) function (needed for multiple vehicle
types)
float DELTA = 0.1; // default small value, later modified in qpx_NET_postOpen()
// for MOVES
float moves[63][41][6]; // moves[regClass][opMode][emissionCategory]
int modeBins[50] = {0}; // opMode distribution

// overall output data
char out1Path[150]; // file "data_sbs.dat"
char out2Path[150]; // file "MOVES_sum.dat"
char out3Path[150]; // file "opMode.dat"
char out4Path[150]; // file "VSP.dat"
char out5Path[150]; // file "TT.dat"
// for equipped vehicle only
```

```
char out11Path[150]; // file "data_sbs_e.dat"
char out21Path[150]; // file "MOVES_sum_e.dat"
char out31Path[150]; // file "opMode_e.dat"
char out41Path[150]; // file "VSP_e.dat"
char out51Path[150]; // file "TT_e.dat"
// for unequipped vehicle only
char out12Path[150]; // file "data_sbs_n.dat"
char out22Path[150]; // file "MOVES_sum_n.dat"
char out32Path[150]; // file "opMode_n.dat"
char out42Path[150]; // file "VSP_n.dat"
char out52Path[150]; // file "TT_n.dat"

float TT_veh[500000]; // travel time of each vehicle

int counter_released = 0;
int counter_arrived = 0;

// aggregated data
// define variables for overall traffic
double VMT = 0, VHT = 0, dist_sum = 0, energy_sum = 0, CO2_sum = 0, CO_sum = 0,
HC_sum = 0, NOx_sum = 0, PM_sum = 0;

//Acceleration/Deceleration Constraints (ECO-CACC)
float max_accel_mpss = 3.5; //3.5 m/s^2 (NOTE: in this case, the selected values
were set to match the selected profile (overall max), other values may be set
(based on speed, or platoon member comfort))
float min_decel_mpss = -7.5; //-7.5 m/s^2

//DEBUG global variables
int DEBUG_FLAG = 0;

//Drawing variables
static float llx = 0.0f, lly = 0.0f, urx = 0.0f, ury = 0.0f;
static Bool rhd;

static void translatePoint(float *x, float *y)
{
	*x = rhd ? *x + llx : *x - llx;
	*y = *y - lly;
}

char *CombinePhases(const char *phase1, const char *phase2)
{
	//Combine two "single NEMA phase" phase strings into one phase string with
the rule G > y > r > R
	int i;
	//char *x = "";//(char *)phase1;
	static char x[13] = "rrrrrrrrrrrr"; //NOTE: remember to use static local
variables when dealing with local pointers being passed out of a function!
	char *output;

	//qps_GUI_printf("%s %d \n", x, strlen(x));

	//int len = strlen(phase1);
```

```
	for (i = 0; i < strlen(phase1); i++)
	{
		if ((phase1[i] == 'G') || (phase2[i] == 'G'))
			x[i] = 'G';
		else if ((phase1[i] == 'y') || (phase2[i] == 'y'))
			x[i] = 'y';
		else if ((phase1[i] == 'r') || (phase2[i] == 'r'))
			x[i] = 'r';
		else //must be "R"
			x[i] = 'R';

		//y[i] = 'G';
	}

	output = x;
	//qps_GUI_printf("%s %d \n", output, strlen(output)); //x is 2x-1 length
instead of x length when the definition x[x length] is used, and x is not
converting back to a char * properly

	return output; //output; //x; //(char *)x;
}

const char *ALLPHASES[49] = {"RrrrrGRrrrrG",
                                                "RrrrryRrrrry",
                                                "RrrrrGRrrrry",
                                                "RrrRrGRrrrrr",
                                                "RrrrryRrrrrG",
                                                "RrrrrrRrrRrG",
                                                "RrrGGGRrrrrr",
                                                "RrryyyRrrrrr",
                                                "RrryyGRrrrrr",
                                                "RrrRrGRrrrrr",
                                                "RrrGGyRrrrrr",
                                                "RrrGGrRrrRrr",
                                                "RrrrrrRrrGGG",
                                                "RrrrrrRrryyy",
                                                "RrrrrrRrrGGy",
                                                "RrrRrrRrrGGr",
                                                "RrrrrrRrryyG",
                                                "RrrrrrRrrRrG",
                                                "RrrGGrRrrGGr",
                                                "RrryyrRrryyr",
                                                "RrryyrRrrGGr",
                                                "RrrRrrRrrGGr",
                                                "RrrGGrRrryyr",
                                                "RrrGGrRrrRrr",
                                                "rrGRrrrrGRrr",
                                                "rryRrrrryRrr",
                                                "rryRrrrrGRrr",
                                                "rrrRrrRrGRrr",
                                                "rrGRrrrryRrr",
                                                "RrGRrrrrrRrr",
                                                "rrrRrrGGGRrr",
                                                "rrrRrryyyRrr",
```

```
                                                        "rrrRrryyGRrr",
                                                        "rrrRrrRrGRrr",
                                                        "rrrRrrGGyRrr",
                                                        "RrrRrrGGrRrr",
                                                        "GGGRrrrrrRrr",
                                                        "yyyRrrrrrRrr",
                                                        "GGyRrrrrrRrr",
                                                        "GGrRrrRrrRrr",
                                                        "yyGRrrrrrRrr",
                                                        "RrGRrrrrrRrr",
                                                        "GGrRrrGGrRrr",
                                                        "yyrRrryyrRrr",
                                                        "GGrRrryyrRrr",
                                                        "GGrRrrRrrRrr",
                                                        "yyrRrrGGrRrr",
                                                        "RrrRrrGGrRrr",
                                                        "rrrrrrrrrrrr"}; //manually copied
from output of print_AllPhases function

//All phase status description
//1) if any light is yellow status = "Y"
//2) if only 0 or 1 phases are green, then status = "R"
//3) if 2 phases are green, then status = "G"
const char *ALLPHASESSTATUS[49] = {"G", "Y", "Y", "R", "Y", "R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "R"};

//The currentPhaseIndex is the row index of NEXTPHASES
int NEXTPHASES[49][8] = {{0, 1, 2, 4},
                                                {48},
                                                {3},
                                                {6},
                                                {5},
                                                {12},
                                                {6, 7, 8, 10},
                                                {48},
                                                {9},
                                                {0},
                                                {11},
                                                {18},
                                                {12, 13, 14, 16},
```

```
                                                {48},
                                                {15},
                                                {18},
                                                {17},
                                                {0},
                                                {18, 19, 20, 22},
                                                {48},
                                                {21},
                                                {12},
                                                {23},
                                                {6},
                                                {24, 25, 26, 28},
                                                {48},
                                                {27},
                                                {30},
                                                {29},
                                                {36},
                                                {30, 31, 32, 34},
                                                {48},
                                                {33},
                                                {24},
                                                {35},
                                                {42},
                                                {36, 37, 38, 40},
                                                {48},
                                                {39},
                                                {42},
                                                {41},
                                                {24},
                                                {42, 43, 44, 46},
                                                {48},
                                                {45},
                                                {36},
                                                {47},
                                                {30},
                                                {0, 6, 12, 18, 24, 30, 36, 42}};
//maybe include 48 here as well (ALLRED can stay ALLRED for more than one second)

//Adaptive/Optimal Signal Control Parameters
int minimumGreenDuration = 8;
int maximumGreenDuration = 65; //set to be slightly larger than the time used for
a queue to clear a link adjacent to the intersection (assumming that the link is
fully loaded when the light turns green, and ignoring overflow into subsequent
links)

int maxGreenReached = 8; // test variable to determine what the longest green
duration during the simulation is

//Adaptive/Optimal Signal Control Variable Declarations and Initializations
float phase_start = 0.0; //in seconds, wrt to paramics time elapsed
int currentPhaseIndex = 0; //default starting phase is ONEFIVEGREEN
int currentPhaseDuration = 8; //= minimumGreenDuration; //default starting
duration is equal to min green (the idea behind a larger min green is to avoid the
partial queue discharge effect)
```

```
VEHICLE* queueTailIDs[16]; //16 lanes for this particular network, change to
accomodate the total number of incoming lanes to the intersection, NOTE: can
change this later to be a property of the intersection, to accomodate multiple
intersections
Bool queuesDischarged = PTRUE; //initialized to true for the sake of the first
phase of the simulation (in which there are no queues),  remember to set and reset
appropriately (set to false at the start of the second phase of the simulation)

//variables for calculating and storing aging factor
int lastPhaseIndex = 0; //phase index of the time step immediately prior to the
current time step, NOTE: can update this as lastPhaseIndex = currentPhaseIndex at
the beginning of each time step in (qpx_CLK_startOfSimLoop())
float timeLastServed[8] = {0}; //absolute time when the individual movement was
last served, NOTE: timeLastServed is absolute time when the movement was last
served (with a green light) and covers movements: 1, 2, 3, ...8
float agingTime[8] = {0}; //size is 8 due to number of individual phases, units is
seconds, NOTE: aging time is time elapsed since the movement was last served and
covers movements: 1, 2, 3, ...8
float agingFactor[8] = {1}; //aging factor muliplies queue length to modulate
fairness, based on elapsed time, NOTE: aging factors refers to dual phases: 1/5,
1/6, 2/5, 2/6, 3/7, 3/8, 4/7, 4/8

//Drawing variables for AdSC
char s0[100], s1[100], s2[100], s3[100], s4[100], s5[100], s6[100], s7[100],
s8[100];
char s9[100];
Bool stringsInitialized = PFALSE; //false if the above strings are not initialized

void print_AllPhases()
{
	//NOTE: the ALLPHASES array contains 49 combinations of individual phases,
if additional phases are desired, the call to this function should be uncommented
	//	This function prints All Phases in the Paramics information
browser.  Copy-paste the text into the ALLPHASES array to complete the procedure.


	//Method 2: works
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN, FIVEGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", ONEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,FIVEGREEN));
	qps_GUI_printf("\"%s\", \n", FIVEGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,SIXGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", ONEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,SIXGREEN));
	qps_GUI_printf("\"%s\", \n", SIXGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,FIVEGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,FIVEYELLOW));
```

```
        qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,FIVEYELLOW));
        qps_GUI_printf("\"%s\", ", TWOGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,FIVEGREEN));
        qps_GUI_printf("\"%s\", \n", FIVEGREEN);

        qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,SIXGREEN));
        qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,SIXYELLOW));
        qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,SIXYELLOW));
        qps_GUI_printf("\"%s\", ", TWOGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,SIXGREEN));
        qps_GUI_printf("\"%s\", \n", SIXGREEN);

        qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,SEVENGREEN));
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,SEVENYELLOW));
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,SEVENYELLOW));
        qps_GUI_printf("\"%s\", ", THREEGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,SEVENGREEN));
        qps_GUI_printf("\"%s\", \n", SEVENGREEN);

        qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,EIGHTGREEN));
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,EIGHTYELLOW));
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,EIGHTYELLOW));
        qps_GUI_printf("\"%s\", ", THREEGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,EIGHTGREEN));
        qps_GUI_printf("\"%s\", \n", EIGHTGREEN);

        qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,SEVENGREEN));
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,SEVENYELLOW));
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,SEVENYELLOW));
        qps_GUI_printf("\"%s\", ", FOURGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,SEVENGREEN));
        qps_GUI_printf("\"%s\", \n", SEVENGREEN);

        qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,EIGHTGREEN));
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,EIGHTYELLOW));
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,EIGHTYELLOW));
        qps_GUI_printf("\"%s\", ", FOURGREEN);
        qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,EIGHTGREEN));
        qps_GUI_printf("\"%s\", \n", EIGHTGREEN);

        qps_GUI_printf("\"%s\" \n", ALLRED);
}

void qpx_NET_postOpen()
{
        int i, j, k, md;
        int reg[13] = {11,21,31,32,41,42,43,51,52,53,54,61,62}; // source types
        char *path, *outPath;
        char inPath[150] = "";
        FILE *fin;

        n[0] = qpg_NET_nodeByIndex(5); //check if this index works
```

```
	WEST_in = qpg_NET_link("11:5"); //this set of link pointers will only work
for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
	NORTH_in = qpg_NET_link("13:5");
	EAST_in = qpg_NET_link("10:5");
	SOUTH_in = qpg_NET_link("12:5");

	WEST_out = qpg_NET_link("5:11");
	NORTH_out = qpg_NET_link("5:13");
	EAST_out = qpg_NET_link("5:10");
	SOUTH_out = qpg_NET_link("5:12");

	//attempt to set dummy phase (only works when links next to zones have
shortened signposts, no idea why, bizarre,...just use the shortened signposts (820
down to 520 ft)
	qps_LNK_priority(NORTH_in, WEST_out, APIPRI_MAJOR);  //4
	qps_LNK_priority(NORTH_in, SOUTH_out, APIPRI_MAJOR); //4
	qps_LNK_priority(NORTH_in, EAST_out, APIPRI_MAJOR);  //7

	qps_LNK_priority(EAST_in, NORTH_out, APIPRI_MAJOR);  //6
	qps_LNK_priority(EAST_in, WEST_out, APIPRI_MAJOR);   //6
	qps_LNK_priority(EAST_in, SOUTH_out, APIPRI_MAJOR);  //1

	qps_LNK_priority(SOUTH_in, EAST_out, APIPRI_MAJOR);  //8
	qps_LNK_priority(SOUTH_in, NORTH_out, APIPRI_MAJOR); //8
	qps_LNK_priority(SOUTH_in, WEST_out, APIPRI_MAJOR);  //3

	qps_LNK_priority(WEST_in, SOUTH_out, APIPRI_MAJOR);  //2
	qps_LNK_priority(WEST_in, EAST_out, APIPRI_MAJOR);  //2
	qps_LNK_priority(WEST_in, NORTH_out, APIPRI_MAJOR); //5



	DELTA = 0.5*qpg_CFG_timeStep();
	path = qpg_NET_dataPath();
	outPath = qpg_NET_statsPath();
	strcpy(inPath, path);
	strcpy(out1Path, outPath);
	strcpy(out2Path, outPath);
	strcpy(out3Path, outPath);
	strcpy(out4Path, outPath);
	strcpy(out5Path, outPath);
	strcpy(out11Path, outPath);
	strcpy(out21Path, outPath);
	strcpy(out31Path, outPath);
	strcpy(out41Path, outPath);
	strcpy(out51Path, outPath);
	strcpy(out12Path, outPath);
	strcpy(out22Path, outPath);
	strcpy(out32Path, outPath);
	strcpy(out42Path, outPath);
	strcpy(out52Path, outPath);

	strcat(inPath, "/sourceTypes_2005.txt");
```

```
	strcat(out1Path, "/data_sbs.dat");
	strcat(out2Path, "/MOVES_sum.dat");
	strcat(out3Path, "/opMode.dat");
	strcat(out4Path, "/VSP.dat");
	strcat(out5Path, "/TT.dat");
	strcat(out11Path, "/data_sbs_e.dat");
	strcat(out21Path, "/MOVES_sum_e.dat");
	strcat(out31Path, "/opMode_e.dat");
	strcat(out41Path, "/VSP_e.dat");
	strcat(out51Path, "/TT_e.dat");
	strcat(out12Path, "/data_sbs_n.dat");
	strcat(out22Path, "/MOVES_sum_n.dat");
	strcat(out32Path, "/opMode_n.dat");
	strcat(out42Path, "/VSP_n.dat");
	strcat(out52Path, "/TT_n.dat");

	fin = fopen(inPath, "r");
	if (fin == NULL)
		qps_GUI_printf("Couldn't open file. (simulation started)");
	fscanf(fin, "%d,", &md); // md: opMode
	for (i = 0; i < 13; i++) // i: regClass
		for (j = 0; j < 41; j++) // j: opMode
			if (md == j)
			{
				fscanf(fin, "%f,%f,%f,%f,%f,%f", &moves[reg[i]][j][0],
&moves[reg[i]][j][1], &moves[reg[i]][j][2], &moves[reg[i]][j][3],
&moves[reg[i]][j][4], &moves[reg[i]][j][5]);
				fscanf(fin, "%d,", &md);
			}
	for (j = 0; j < 41; j++) {
		for (k = 0; k <= 5; k++) {
			moves[20][j][k] = moves[21][j][k];
		}
	}
	fclose(fin);

	//Drawing initializations
	/** These are set here so as to optimise the drawing code. */

	/* Obtain the bounding box of the network. If the point (0,0) is passed to
	* qps_DRW_filledCircle, then the circle will be drawn at (llx, lly)....
	*/
	qpg_POS_network(&llx, &lly, &urx, &ury);

	/**
	* We need to correct x coordinates if the network is right hand drive.
	*/
	rhd = qpg_CFG_driveOnRight(); //this should return true;
	qps_GUI_printf("NET_postOpen entered\n");
}

void qpx_DRW_modelView()
{
	//Adaptive/Optimal Signal Control Test Drawing Code
```

```
	float currentTime = qpg_CFG_simulationTime(); //elapsed simulation time in
seconds
	//test variables
	//int queue_length = 0;
	//drawing variables
	float circleCentreX = -113.36f, circleCentreY = 631.52f; //meters
	float rectangleCenterX = 173.36f, rectangleCenterY = 548.64f; //meters
	float yOffset = 0;
	//char s0[100], s1[100], s2[100], s3[100], s4[100], s5[100], s6[100],
s7[100], s8[100]; //maybe extend length, that worked, NOTE:can make this into a 2d
array later if there are lots of strings to draw
	float testSpeed = 60;
	VEHICLE* temp_queueTailIDs[16];

	//update data once a second (to prevent large overhead)
	if (fabs(currentTime - (int)currentTime) < DELTA)
	{
		//set strings
		/*sprintf(s0, "Combined Queue Lengths by NEMA Phase: ");
		sprintf(s1, "1/5: %d ", Find_Phase_Queue_Length(5, ALLPHASES[0],
&temp_queueTailIDs));
		sprintf(s2, "1/6: %d ", Find_Phase_Queue_Length(5, ALLPHASES[6],
&temp_queueTailIDs));
		sprintf(s3, "2/5: %d ", Find_Phase_Queue_Length(5, ALLPHASES[12],
&temp_queueTailIDs));
		sprintf(s4, "2/6: %d ", Find_Phase_Queue_Length(5, ALLPHASES[18],
&temp_queueTailIDs));
		sprintf(s5, "3/7: %d ", Find_Phase_Queue_Length(5, ALLPHASES[24],
&temp_queueTailIDs));
		sprintf(s6, "3/8: %d ", Find_Phase_Queue_Length(5, ALLPHASES[30],
&temp_queueTailIDs));
		sprintf(s7, "4/7: %d ", Find_Phase_Queue_Length(5, ALLPHASES[36],
&temp_queueTailIDs));
		sprintf(s8, "4/8: %d ", Find_Phase_Queue_Length(5, ALLPHASES[42],
&temp_queueTailIDs));*/

		sprintf(s0, "Combined Queue Lengths by NEMA Phase, QL x
AgingFactor");
		sprintf(s1, "1/5: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[0], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[0],
&temp_queueTailIDs)*agingFactor[0]);
		sprintf(s2, "1/6: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[6], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[6],
&temp_queueTailIDs)*agingFactor[1]);
		sprintf(s3, "2/5: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[12], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[12],
&temp_queueTailIDs)*agingFactor[2]);
		sprintf(s4, "2/6: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[18], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[18],
&temp_queueTailIDs)*agingFactor[3]);
		sprintf(s5, "3/7: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[24], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[24],
&temp_queueTailIDs)*agingFactor[4]);
```

```
		sprintf(s6, "3/8: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[30], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[30],
&temp_queueTailIDs)*agingFactor[5]);
		sprintf(s7, "4/7: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[36], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[36],
&temp_queueTailIDs)*agingFactor[6]);
		sprintf(s8, "4/8: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[42], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[42],
&temp_queueTailIDs)*agingFactor[7]);

		sprintf(s9, "Maximum Green Reached: %d ", maxGreenReached);

		stringsInitialized = PTRUE;
	}
	else if (stringsInitialized == PFALSE) //then initialized strings with 0
values for queue lengths
	{
		//set strings
		/*sprintf(s0, "Combined Queue Lengths by NEMA Phase: ");
		sprintf(s1, "1/5: %d ", 0);
		sprintf(s2, "1/6: %d ", 0);
		sprintf(s3, "2/5: %d ", 0);
		sprintf(s4, "2/6: %d ", 0);
		sprintf(s5, "3/7: %d ", 0);
		sprintf(s6, "3/8: %d ", 0);
		sprintf(s7, "4/7: %d ", 0);
		sprintf(s8, "4/8: %d ", 0);*/

		sprintf(s0, "Combined Queue Lengths by NEMA Phase, QL x
AgingFactor");
		sprintf(s1, "1/5: %3d,  %5.2f ", 0, 0);
		sprintf(s2, "1/6: %3d,  %5.2f ", 0, 0);
		sprintf(s3, "2/5: %3d,  %5.2f ", 0, 0);
		sprintf(s4, "2/6: %3d,  %5.2f ", 0, 0);
		sprintf(s5, "3/7: %3d,  %5.2f ", 0, 0);
		sprintf(s6, "3/8: %3d,  %5.2f ", 0, 0);
		sprintf(s7, "4/7: %3d,  %5.2f ", 0, 0);
		sprintf(s8, "4/8: %3d,  %5.2f ", 0, 0);

		sprintf(s9, "Maximum Green Reached: %d ", 0);

		stringsInitialized = PTRUE;
	}
	//translatePoint(&circleCentreX, &circleCentreY);
	if (currentTime > 1.5)
	{
		//NOTE: the following drawing code is called once every time step
(otherwise the code will not be displayed
		/**
		* Translate the coordinates of the circle centre, and correct for
right hand drive (if needed).
		*/
		translatePoint(&circleCentreX, &circleCentreY);
		translatePoint(&rectangleCenterX, &rectangleCenterY);
```

```
            qps_DRW_colourRGB(1, 1, 1);
            //NOTE: x is flipped
            qps_DRW_stringXY(s0, circleCentreX + 30.0, circleCentreY + 5.0,
10.0); //first draw string, NOTE: text height is in meters
            qps_DRW_stringXY(s1, circleCentreX + 30.0, circleCentreY - 5.0,
10.0);
            qps_DRW_stringXY(s2, circleCentreX + 30.0, circleCentreY - 15.0,
10.0);
            qps_DRW_stringXY(s3, circleCentreX + 30.0, circleCentreY - 25.0,
10.0);
            qps_DRW_stringXY(s4, circleCentreX + 30.0, circleCentreY - 35.0,
10.0);
            qps_DRW_stringXY(s5, circleCentreX + 30.0, circleCentreY - 45.0,
10.0);
            qps_DRW_stringXY(s6, circleCentreX + 30.0, circleCentreY - 55.0,
10.0);
            qps_DRW_stringXY(s7, circleCentreX + 30.0, circleCentreY - 65.0,
10.0);
            qps_DRW_stringXY(s8, circleCentreX + 30.0, circleCentreY - 75.0,
10.0);

            qps_DRW_stringXY(s9, rectangleCenterX, rectangleCenterY, 10.0);

            //Draw the circle
            qps_DRW_colourRGB(0, 1, 0);
            //set position of circle depending on active phase
            if (currentPhaseIndex == 0)
            {
                  yOffset = -5.0;
            }
            else if (currentPhaseIndex == 6)
            {
                  yOffset = -15.0;
            }
            else if (currentPhaseIndex == 12)
            {
                  yOffset = -25.0;
            }
            else if (currentPhaseIndex == 18)
            {
                  yOffset = -35.0;
            }
            else if (currentPhaseIndex == 24)
            {
                  yOffset = -45.0;
            }
            else if (currentPhaseIndex == 30)
            {
                  yOffset = -55.0;
            }
            else if (currentPhaseIndex == 36)
            {
                  yOffset = -65.0;
```

```
		}
		else if (currentPhaseIndex == 42)
		{
			yOffset = -75.0;
		}

		if (ALLPHASESSTATUS[currentPhaseIndex] == "G")
		{
			qps_DRW_filledCircle(circleCentreX + 40.0, circleCentreY +
yOffset + 5.0, 0, 5); //then draw circle, NOTE: radius is in meters
		}

		//Draw the rectangle
		qps_DRW_colourRGB(1, 0, 0);
		qps_DRW_filledRectangleXY(rectangleCenterX + 2.5, rectangleCenterY -
2.5, rectangleCenterX - 140.0, rectangleCenterY + 12.5); //float bl_x, float bl_y,
float tr_x, float tr_y

		//qps_GUI_printf("DRW_modelView entered\n");
	}
}

int opMode(float vsp, float speed, float acc)
{
	speed = speed*INDEX;
	acc = acc*INDEX;
	if (acc <= -2) return 0;

	if (speed < 1 && speed >= -1) return 1;
	if (speed < 25 && speed >= 0) //this second value should be 1, (error in
Haitao's code, never reached though, due to previous statement)
	{
		if (vsp < 0) return 11;
		if (vsp < 3) return 12;
		if (vsp < 6) return 13;
		if (vsp < 9) return 14;
		if (vsp < 12) return 15;
		return 16;}
	else if (speed < 50) {
		if (vsp < 0) return 21;
		if (vsp < 3) return 22;
		if (vsp < 6) return 23;
		if (vsp < 9) return 24;
		if (vsp < 12) return 25;
		if (vsp < 18) return 27;
		if (vsp < 24) return 28;
		if (vsp < 30) return 29;
		return 30;}
	else {
		if (vsp < 6) return 33;
		if (vsp < 12) return 35;
		if (vsp < 18) return 37;
		if (vsp < 24) return 38;
		if (vsp < 30) return 39;
```

```
                return 40;}
}

void updateVehAttributes(VEHICLE_DATA* myVeh)
{
        if(myVeh->sourceType == 11)
        {
                myVeh->A = 0.0251;
         myVeh->B = 0;
         myVeh->C = 0.000315;
         myVeh->M = 0.285;
                myVeh->f = 0.285;
                //myVeh->regClass = 10;
        }
        else if(myVeh->sourceType == 21 || myVeh->sourceType == 20)
        {
                myVeh->A = 0.156461;
         myVeh->B = 0.002002;
         myVeh->C = 0.000493;
         myVeh->M = 1.4788;
                myVeh->f = 1.4788;
                //myVeh->regClass = 20;
        }
        else if(myVeh->sourceType == 31)
        {
                myVeh->A = 0.22112;
         myVeh->B = 0.002838;
         myVeh->C = 0.000698;
         myVeh->M = 1.86686;
                myVeh->f = 1.86686;
                //myVeh->regClass = 30;
        }
        else if(myVeh->sourceType == 32)
        {
                myVeh->A = 0.235008;
         myVeh->B = 0.003039;
         myVeh->C = 0.000748;
         myVeh->M = 2.05979;
                myVeh->f = 2.05979;
                //myVeh->regClass = 30;
        }
        else if(myVeh->sourceType == 41)
        {
                myVeh->A = 1.29515;
         myVeh->B = 0;
         myVeh->C = 0.003715;
         myVeh->M = 19.5937;
                myVeh->f = 17.1;
                //myVeh->regClass = 48;
        }
        else if(myVeh->sourceType == 42)
        {
                myVeh->A = 1.0944;
         myVeh->B = 0;
```

```
 myVeh->C = 0.003587;
 myVeh->M = 16.556;
       myVeh->f = 17.1;
       //myVeh->regClass = 48;
}
else if(myVeh->sourceType == 43)
{
       myVeh->A = 0.746718;
 myVeh->B = 0;
 myVeh->C = 0.002176;
 myVeh->M = 9.06989;
       myVeh->f = 17.1;
       //myVeh->regClass = 46;
}
else if(myVeh->sourceType == 51)
{
       myVeh->A = 1.41705;
 myVeh->B = 0;
 myVeh->C = 0.003572;
 myVeh->M = 20.6845;
       myVeh->f = 17.1;
       //myVeh->regClass = 47;
}
else if(myVeh->sourceType == 52)
{
       myVeh->A = 0.561933;
 myVeh->B = 0;
 myVeh->C = 0.001603;
 myVeh->M = 7.64159;
       myVeh->f = 17.1;
       //myVeh->regClass = 42;
}
else if(myVeh->sourceType == 53)
{
       myVeh->A = 0.498699;
 myVeh->B = 0;
 myVeh->C = 0.001474;
 myVeh->M = 6.25047;
       myVeh->f = 17.1;
       //myVeh->regClass = 41;
}
else if(myVeh->sourceType == 54)
{
       myVeh->A = 0.617371;
 myVeh->B = 0;
 myVeh->C = 0.002105;
 myVeh->M = 6.73483;
       myVeh->f = 17.1;
       //myVeh->regClass = 42;
}
else if(myVeh->sourceType == 61)
{
       myVeh->A = 1.96354;
 myVeh->B = 0;
```

```
        myVeh->C = 0.004031;
        myVeh->M = 29.3275;
              myVeh->f = 17.1;
              //myVeh->regClass = 47;
       }
       else //if(myVeh->sourceType == 62)
       {
              myVeh->A = 2.08126;
        myVeh->B = 0;
        myVeh->C = 0.004188;
        myVeh->M = 31.4038;
              myVeh->f = 17.1;
              //myVeh->regClass = 47;
       }
}

void  qpx_VHC_release(VEHICLE* Vp)
{
       VEHICLE_DATA *myVeh = calloc(1, sizeof(VEHICLE_DATA));
       double rn;
       float x5 = 0; // used for destination node location
       float y5 = 0; // used for destination node location
       float z5 = 0; //unused
       LINK* link = qpg_VHC_link(Vp);           // pointer of the current link
(used for ESH)
       counter_released++;

       // initialize data for MOVES model
       myVeh->ID = qpg_VHC_uniqueID(Vp);
       myVeh->sourceType = qpg_VHC_type(Vp); // emission tables directly related
to source types
       myVeh->theta = 0;
       myVeh->first2sec = 1;
       myVeh->origin = qpg_VHC_origin(Vp);
       myVeh->dest = qpg_VHC_destination(Vp);
       myVeh->tripDist = 0;//qpg_RTR_distanceRemaining(qpg_VHC_link(Vp), Vp);
//this function is bugged, and is not working with the on/off ramp network

       //retrieve destination node position (for CACC)
       qpg_POS_node(qpg_LNK_nodeStart(qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1)),
&x5, &y5, &z5);
       myVeh->x_destNode = x5;
       myVeh->y_destNode = y5;

       myVeh->total_energy = 0; //default initialization, aggregated when vehicle
is in the network

       updateVehAttributes(myVeh);
       qps_VHC_userdata(Vp, (VEHICLE_DATA *)myVeh);

       //qps_GUI_printf("Vehicle %d trip %d -> %d on link %s\n",
qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp), qpg_VHC_destination(Vp),
qpg_LNK_name(qpg_VHC_link(Vp)));
```

```
	}

void calEmissions(VEHICLE_DATA* myVeh, float currTime, float vel_c, int linkid)
{
	int i;
	// shift 4-second data
	for (i = 0; i < 3; i++)
	{
		myVeh->vel[i] = myVeh->vel[i+1];
		myVeh->acc[i] = myVeh->acc[i+1];
		myVeh->linkID[i] = myVeh->linkID[i+1];
		myVeh->VSP[i] = myVeh->VSP[i+1];
		myVeh->mode[i] = myVeh->mode[i+1];
		myVeh->tm[i] = myVeh->tm[i+1];
		myVeh->energy[i] = myVeh->energy[i+1];
		myVeh->CO2[i] = myVeh->CO2[i+1];
		myVeh->CO[i] = myVeh->CO[i+1];
		myVeh->HC[i] = myVeh->HC[i+1];
		myVeh->NOx[i] = myVeh->NOx[i+1];
		myVeh->PM[i] = myVeh->PM[i+1];
	}
	myVeh->tm[3] = currTime;
	myVeh->linkID[3] = linkid;
	myVeh->vel[3] = vel_c;
	myVeh->acc[2] = (myVeh->vel[3] - myVeh->vel[1])/2;
	myVeh->VSP[2] = (myVeh->A*myVeh->vel[2] + myVeh->B*pow(myVeh->vel[2],2) +
myVeh->C*pow(myVeh->vel[2],3) + myVeh->M*(myVeh->acc[2] + GRAV*sin(myVeh-
>theta))*myVeh->vel[2])/myVeh->f;
	myVeh->mode[2] = opMode(myVeh->VSP[2], myVeh->vel[2], myVeh->acc[2]);
	if (myVeh->acc[0] < -1 && myVeh->acc[1] < -1 && myVeh->acc[2] < -1) myVeh-
>mode[2] = 0;
	modeBins[myVeh->mode[2]] += 1;
	myVeh->HC[2] = moves[myVeh->sourceType][myVeh->mode[2]][0];
	myVeh->CO[2] = moves[myVeh->sourceType][myVeh->mode[2]][1];
	myVeh->NOx[2] = moves[myVeh->sourceType][myVeh->mode[2]][2];
	myVeh->CO2[2] = moves[myVeh->sourceType][myVeh->mode[2]][3];
	myVeh->energy[2] = moves[myVeh->sourceType][myVeh->mode[2]][4];
	myVeh->PM[2] = moves[myVeh->sourceType][myVeh->mode[2]][5];

	//update overall statistics
	dist_sum += myVeh->vel[2];
	energy_sum += myVeh->energy[2];
	CO2_sum += myVeh->CO2[2];
	CO_sum += myVeh->CO[2];
	HC_sum += myVeh->HC[2];
	NOx_sum += myVeh->NOx[2];
	PM_sum += myVeh->PM[2];

	//aggregate individual vehicle energy
	myVeh->total_energy += myVeh->energy[2];
	myVeh->total_dist += myVeh->vel[2];
}

void SetRedYellowGreenState(int intersectionID, const char *phaseString)
```

```
{
	//This function sets the priorities of a 4-way intersection to mimic
adaptive signal control
	//Adaptive Signal Control Work-around:
	// MAJOR = Green    (G)
	// MEDIUM = Yellow  (y)
	// MINOR = RTOR     (R)
	// BARRED = Red     (r)

	int i;
	int priorityArray[12]; //holds priority integers (based on paramics API: 0
= Major, 1 = Medium, 2 = Minor, 3 = Barred) for all 12 movements of a 4-way
intersection
	//declare pointers (these are the links connected to the junction specified
by the intersectionID parameter)
	LINK* WEST_in_local; //check if this works
	LINK* NORTH_in_local;
	LINK* EAST_in_local;
	LINK* SOUTH_in_local;

	LINK* WEST_out_local;
	LINK* NORTH_out_local;
	LINK* EAST_out_local;
	LINK* SOUTH_out_local;

	//NOTE: can use intersectionID later to add additional intersections
	//NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

	//METHOD 1: manual specification of links connected to junction //NOTE: can
later add a function to programatically find these links (in the case of multiple
junctions)
	WEST_in_local = qpg_NET_link("11:5"); //this set of link pointers will only
work for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
	NORTH_in_local = qpg_NET_link("13:5");
	EAST_in_local = qpg_NET_link("10:5");
	SOUTH_in_local = qpg_NET_link("12:5");

	WEST_out_local = qpg_NET_link("5:11");
	NORTH_out_local = qpg_NET_link("5:13");
	EAST_out_local = qpg_NET_link("5:10");
	SOUTH_out_local = qpg_NET_link("5:12");

	//translate phaseString into priority array
	for (i = 0; i < strlen(phaseString); i++) //length of phaseString should be
12
	{
		if (phaseString[i] == 'G') //double-check the quotes
		{
			priorityArray[i] = APIPRI_MAJOR; //mimics green light
		}
		else if (phaseString[i] == 'y')
		{
```

```
                        priorityArray[i] = APIPRI_MEDIUM; // mimics yellow light
                }
                else if (phaseString[i] == 'R')
                {
                        priorityArray[i] = APIPRI_MINOR; // mimics RTOR (right turn on
red)
                }
                else if (phaseString[i] == 'r')
                {
                        priorityArray[i] = APIPRI_BARRED; // mimics red light
                }
        }

        //set priorities                                          //NEMA phase
equivalent
        qps_LNK_priority(NORTH_in, WEST_out, priorityArray[0]);  //4
        qps_LNK_priority(NORTH_in, SOUTH_out, priorityArray[1]); //4
        qps_LNK_priority(NORTH_in, EAST_out, priorityArray[2]);  //7

        qps_LNK_priority(EAST_in, NORTH_out, priorityArray[3]);  //6
        qps_LNK_priority(EAST_in, WEST_out, priorityArray[4]);   //6
        qps_LNK_priority(EAST_in, SOUTH_out, priorityArray[5]);  //1

        qps_LNK_priority(SOUTH_in, EAST_out, priorityArray[6]);  //8
        qps_LNK_priority(SOUTH_in, NORTH_out, priorityArray[7]); //8
        qps_LNK_priority(SOUTH_in, WEST_out, priorityArray[8]);  //3

        qps_LNK_priority(WEST_in, SOUTH_out, priorityArray[9]);  //2
        qps_LNK_priority(WEST_in, EAST_out, priorityArray[10]);  //2
        qps_LNK_priority(WEST_in, NORTH_out, priorityArray[11]); //5

}

VEHICLE *GetQueueTailVp(LINK *incomingLink, int incomingLane, int
lane_queue_length1, LINK *permittedNextLink1, LINK *permittedNextLink2)
{
        //Finds the pointer of the first vehicle (starting from the rear of the
queue which has a correct next movement for the given lane
        //If no such vehicle is found, return value is NULL
        //parameter description:
        //     incomingLink: is the pointer to one of the links that permits
vehicles to enter the intersection
        //  incomingLane: is the lane index of the lane currently being examined
        //  lane_queue_length: is the number of vehicles currently on the lane-link
combination
        //  permittedNextLink#: refers to permitted next links based on the current
lane-link
        //            NOTE: currently this function supports up to two next links,
(2 next links indicates a shared movement)
        //                     If there is only one next link (no shared movement),
then the remaining next link pointers should be set to NULL
        VEHICLE *tempVp = qpg_LNK_vehicleTail(incomingLink, incomingLane); //used
in the process of finding the vehicle pointer of the correct vehicle tail
```

```
	int temp_lane_queue_length = lane_queue_length1; //used in the case where
the actual tail vehicle of the platoon is not in the correct lane for its intended
movement through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found

	if (lane_queue_length1 > 0)
	{
		if (permittedNextLink2 != NULL)
		{
			if ((qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink1) || (qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink2)) //shared movement
			{
				//out_temp_queueTailIDs[0] =
qpg_LNK_vehicleTail(NORTH_in_local, 1);
				return tempVp;
			}
			else if (temp_lane_queue_length == 1) //only one vehicle was
in the lane, use dummy queueTailID bc vehicle is in wrong lane
			{
				//out_temp_queueTailIDs[0] = NULL; //check if this
works
				return NULL;
			}
			else //temp_lane_queue_length is >= 2 and actual queue tail
has wrong lane
			{
				while ((temp_lane_queue_length > 1) && (FOUND ==
PFALSE))
				{
					temp_lane_queue_length -= 1;
					tempVp = qpg_VHC_ahead(tempVp); //get the next
vehicle ahead in the queue
					if ((qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink1) || (qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink2)) //shared movement
					{
						//out_temp_queueTailIDs[0] = tempVp;
						return tempVp;
						FOUND = PTRUE; //unnecessary
					}
				}

				if (FOUND == PFALSE) //temp_lane_queue_length should =
1 at this point, if FOUND == PFALSE
				{
					//out_temp_queueTailIDs[0] = NULL; //check if
this works
					return NULL;
				}

			}
		}
```

```
		else //no shared movement, (only one next link is permitted)
		{
			if (qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink1)
			{
				//out_temp_queueTailIDs[0] =
qpg_LNK_vehicleTail(NORTH_in_local, 1);
				return tempVp;
			}
			else if (temp_lane_queue_length == 1) //only one vehicle was
in the lane, use dummy queueTailID bc vehicle is in wrong lane
			{
				//out_temp_queueTailIDs[0] = NULL; //check if this
works
				return NULL;
			}
			else //temp_lane_queue_length is >= 2 and actual queue tail
has wrong lane
			{
				while ((temp_lane_queue_length > 1) && (FOUND ==
PFALSE))
				{
					temp_lane_queue_length -= 1;
					tempVp = qpg_VHC_ahead(tempVp); //get the next
vehicle ahead in the queue
					if (qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink1)
					{
						//out_temp_queueTailIDs[0] = tempVp;
						return tempVp;
						FOUND = PTRUE; //unnecessary
					}
				}

				if (FOUND == PFALSE) //temp_lane_queue_length should =
1 at this point, if FOUND == PFALSE
				{
					//out_temp_queueTailIDs[0] = NULL; //check if
this works
					return NULL;
				}

			}
		}
	}
	else
	{
		return NULL;
	}
}

int *Find_All_Individual_Phase_Queue_Lengths(int intersectionID)
{
```

```
	//This function finds the total queue length for a given individual phase
(i.e. half of the NEMA dual phase), summing up over all lanes belonging to the
movement
	//NOTE: This function simply counts all vehicles on the lane-link closest
to intersection regardless of range or speed
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve
	static int individual_phase_queue_length[8] = {0};
	int *output;

	//declare pointers (these are the links connected to the junction specified
by the intersectionID parameter)
	LINK* WEST_in_local; //check if this works
	LINK* NORTH_in_local;
	LINK* EAST_in_local;
	LINK* SOUTH_in_local;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local;
	LINK* NORTH_out_local;
	LINK* EAST_out_local;
	LINK* SOUTH_out_local;

	//NOTE: can use intersectionID later to add additional intersections
	//NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

	//METHOD 1: manual specification of links connected to junction //NOTE: can
later add a function to programatically find these links (in the case of multiple
junctions)
	WEST_in_local = qpg_NET_link("11:5"); //this set of link pointers will only
work for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
	NORTH_in_local = qpg_NET_link("13:5");
	EAST_in_local = qpg_NET_link("10:5");
	SOUTH_in_local = qpg_NET_link("12:5");

	WEST_out_local = qpg_NET_link("5:11");
	NORTH_out_local = qpg_NET_link("5:13");
	EAST_out_local = qpg_NET_link("5:10");
	SOUTH_out_local = qpg_NET_link("5:12");

	//currentQueueIDList = [] #initialize list as empty at the beginning of
each function call

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate combined queue length: sum over all lanes of the
prospective phase to see how many vehicles would be served by activating this
phase
```

```
	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)

	//1)Get individual lane queue lengths
	// NEMA Phase 4
	lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
	lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
	lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);

	// NEMA Phase 7
	lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);

	// NEMA Phase 6
	lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
	lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
	lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);

	// NEMA Phase 1
	lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);

	// NEMA Phase 8
	lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
	lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
	lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);

	// NEMA Phase 3
	lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);

	// NEMA Phase 2
	lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
	lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
	lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);

	// NEMA Phase 5
	lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);

	//2) aggregate lanes for individual movements
	individual_phase_queue_length[0] = lane_queue_length[7]; // NEMA Phase 1
	individual_phase_queue_length[1] = lane_queue_length[12] +
lane_queue_length[13] + lane_queue_length[14]; // NEMA Phase 2
	individual_phase_queue_length[2] = lane_queue_length[11]; // NEMA Phase 3
	individual_phase_queue_length[3] = lane_queue_length[0] +
lane_queue_length[1] + lane_queue_length[2]; // NEMA Phase 4
	individual_phase_queue_length[4] = lane_queue_length[15]; // NEMA Phase 5
	individual_phase_queue_length[5] = lane_queue_length[4] +
lane_queue_length[5] + lane_queue_length[6]; // NEMA Phase 6
	individual_phase_queue_length[6] = lane_queue_length[3]; // NEMA Phase 7
	individual_phase_queue_length[7] = lane_queue_length[8] +
lane_queue_length[9] + lane_queue_length[10]; // NEMA Phase 8

	output = individual_phase_queue_length;
	return individual_phase_queue_length; //output;
}
```

```
int Find_Phase_Queue_Length(int intersectionID, char *phaseString, VEHICLE*
out_temp_queueTailIDs[16])
{
	//This function finds the total queue length for a given phase, summing up
all lanes and specified phases
	//NOTE: This function simply counts all vehicles on the lane-link closest
to intersection regardless of range or speed
	int total_queue_length = 0;
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve

	VEHICLE *tempVp; //used in the process of finding the vehicle pointer of
the correct vehicle tail
	int temp_lane_queue_length = 0; //used in the case where the actual tail
vehicle of the platoon is not in the correct lane for its intended movement
through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found

	//declare pointers (these are the links connected to the junction specified
by the intersectionID parameter)
	LINK* WEST_in_local; //check if this works
	LINK* NORTH_in_local;
	LINK* EAST_in_local;
	LINK* SOUTH_in_local;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local;
	LINK* NORTH_out_local;
	LINK* EAST_out_local;
	LINK* SOUTH_out_local;

	//NOTE: can use intersectionID later to add additional intersections
	//NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

	//METHOD 1: manual specification of links connected to junction //NOTE: can
later add a function to programatically find these links (in the case of multiple
junctions)
	WEST_in_local = qpg_NET_link("11:5"); //this set of link pointers will only
work for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
	NORTH_in_local = qpg_NET_link("13:5");
	EAST_in_local = qpg_NET_link("10:5");
	SOUTH_in_local = qpg_NET_link("12:5");

	WEST_out_local = qpg_NET_link("5:11");
	NORTH_out_local = qpg_NET_link("5:13");
	EAST_out_local = qpg_NET_link("5:10");
	SOUTH_out_local = qpg_NET_link("5:12");

	//currentQueueIDList = [] #initialize list as empty at the beginning of
each function call
```

```
	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
		lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);
		total_queue_length += lane_queue_length[0] + lane_queue_length[1] +
lane_queue_length[2];

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[0] = GetQueueTailVp(NORTH_in_local, 1,
lane_queue_length[0], WEST_out_local, SOUTH_out_local);//(LINK *incomingLink, int
incomingLane, int lane_queue_length, LINK *permittedNextLink1, LINK
*permittedNextLink2);


		//out_temp_queueTailIDs[1] = qpg_LNK_vehicleTail(NORTH_in_local, 2);
		out_temp_queueTailIDs[1] = GetQueueTailVp(NORTH_in_local, 2,
lane_queue_length[1], SOUTH_out_local, NULL);
		//out_temp_queueTailIDs[2] = qpg_LNK_vehicleTail(NORTH_in_local, 3);
		out_temp_queueTailIDs[2] = GetQueueTailVp(NORTH_in_local, 3,
lane_queue_length[2], SOUTH_out_local, NULL);
	}
	if (phaseString[2] == 'G')
	{
		lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);
		total_queue_length += lane_queue_length[3];

		out_temp_queueTailIDs[3] = GetQueueTailVp(NORTH_in_local, 4,
lane_queue_length[3], EAST_out_local, NULL);//qpg_LNK_vehicleTail(NORTH_in_local,
4);
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
		lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
		lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);
```

```
		total_queue_length += lane_queue_length[4] + lane_queue_length[5] +
lane_queue_length[6];

		out_temp_queueTailIDs[4] = GetQueueTailVp(EAST_in_local, 1,
lane_queue_length[4], NORTH_out_local,
WEST_out_local);//qpg_LNK_vehicleTail(EAST_in_local, 1);
		out_temp_queueTailIDs[5] = GetQueueTailVp(EAST_in_local, 2,
lane_queue_length[5], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
2);
		out_temp_queueTailIDs[6] = GetQueueTailVp(EAST_in_local, 3,
lane_queue_length[6], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
3);
	}
	if (phaseString[5] == 'G')
	{
		lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);
		total_queue_length += lane_queue_length[7];

		out_temp_queueTailIDs[7] = GetQueueTailVp(EAST_in_local, 4,
lane_queue_length[7], SOUTH_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
4);
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
		lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
		lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);
		total_queue_length += lane_queue_length[8] + lane_queue_length[9] +
lane_queue_length[10];

		out_temp_queueTailIDs[8] = GetQueueTailVp(SOUTH_in_local, 1,
lane_queue_length[8], EAST_out_local,
NORTH_out_local);//qpg_LNK_vehicleTail(SOUTH_in_local, 1);
		out_temp_queueTailIDs[9] = GetQueueTailVp(SOUTH_in_local, 2,
lane_queue_length[9], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
2);
		out_temp_queueTailIDs[10] = GetQueueTailVp(SOUTH_in_local, 3,
lane_queue_length[10], NORTH_out_local,
NULL);//qpg_LNK_vehicleTail(SOUTH_in_local, 3);
	}
	if (phaseString[8] == 'G')
	{
		lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);
		total_queue_length += lane_queue_length[11];

		out_temp_queueTailIDs[11] = GetQueueTailVp(SOUTH_in_local, 4,
lane_queue_length[11], WEST_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
4);
	}
```

```
	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
		lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
		lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);
		total_queue_length += lane_queue_length[12] + lane_queue_length[13]
+ lane_queue_length[14];

		out_temp_queueTailIDs[12] = GetQueueTailVp(WEST_in_local, 1,
lane_queue_length[12], SOUTH_out_local,
EAST_out_local);//qpg_LNK_vehicleTail(WEST_in_local, 1);
		out_temp_queueTailIDs[13] = GetQueueTailVp(WEST_in_local, 2,
lane_queue_length[13], EAST_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
2);

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[14] = GetQueueTailVp(WEST_in_local, 3,
lane_queue_length[14], EAST_out_local, NULL);

	}
	if (phaseString[11] == 'G')
	{
		lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);
		total_queue_length += lane_queue_length[15];

		out_temp_queueTailIDs[15] = GetQueueTailVp(WEST_in_local, 4,
lane_queue_length[15], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
4);
	}
	return total_queue_length;
}

int Find_Phase_Queue_Length2(int intersectionID, char *phaseString, VEHICLE*
out_temp_queueTailIDs[16])
{
	//This function finds the total queue length for a given phase, summing up
all lanes and specified phases
	//NOTE: This function simply counts all queued vehicles on the lane-link
closest to intersection that are under a maximum speed threshold (Is this really
worth the effort?)
	int total_queue_length = 0;
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve

	VEHICLE *tempVp; //used in the process of finding the vehicle pointer of
the correct vehicle tail
	int temp_lane_queue_length = 0; //used in the case where the actual tail
vehicle of the platoon is not in the correct lane for its intended movement
through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found
```

```
	//declare pointers (these are the links connected to the junction specified
by the intersectionID parameter)
	LINK* WEST_in_local; //check if this works
	LINK* NORTH_in_local;
	LINK* EAST_in_local;
	LINK* SOUTH_in_local;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local;
	LINK* NORTH_out_local;
	LINK* EAST_out_local;
	LINK* SOUTH_out_local;

	//NOTE: can use intersectionID later to add additional intersections
	//NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

	//METHOD 1: manual specification of links connected to junction //NOTE: can
later add a function to programatically find these links (in the case of multiple
junctions)
	WEST_in_local = qpg_NET_link("11:5"); //this set of link pointers will only
work for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
	NORTH_in_local = qpg_NET_link("13:5");
	EAST_in_local = qpg_NET_link("10:5");
	SOUTH_in_local = qpg_NET_link("12:5");

	WEST_out_local = qpg_NET_link("5:11");
	NORTH_out_local = qpg_NET_link("5:13");
	EAST_out_local = qpg_NET_link("5:10");
	SOUTH_out_local = qpg_NET_link("5:12");

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
		lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);
		total_queue_length += lane_queue_length[0] + lane_queue_length[1] +
lane_queue_length[2];
```

```
		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[0] = GetQueueTailVp(NORTH_in_local, 1,
lane_queue_length[0], WEST_out_local, SOUTH_out_local);//(LINK *incomingLink, int
incomingLane, int lane_queue_length, LINK *permittedNextLink1, LINK
*permittedNextLink2);
		out_temp_queueTailIDs[1] = GetQueueTailVp(NORTH_in_local, 2,
lane_queue_length[1], SOUTH_out_local, NULL);
		out_temp_queueTailIDs[2] = GetQueueTailVp(NORTH_in_local, 3,
lane_queue_length[2], SOUTH_out_local, NULL);
	}
	if (phaseString[2] == 'G')
	{
		lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);
		total_queue_length += lane_queue_length[3];

		out_temp_queueTailIDs[3] = GetQueueTailVp(NORTH_in_local, 4,
lane_queue_length[3], EAST_out_local, NULL);//qpg_LNK_vehicleTail(NORTH_in_local,
4);
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
		lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
		lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);
		total_queue_length += lane_queue_length[4] + lane_queue_length[5] +
lane_queue_length[6];

		out_temp_queueTailIDs[4] = GetQueueTailVp(EAST_in_local, 1,
lane_queue_length[4], NORTH_out_local,
WEST_out_local);//qpg_LNK_vehicleTail(EAST_in_local, 1);
		out_temp_queueTailIDs[5] = GetQueueTailVp(EAST_in_local, 2,
lane_queue_length[5], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
2);
		out_temp_queueTailIDs[6] = GetQueueTailVp(EAST_in_local, 3,
lane_queue_length[6], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
3);
	}
	if (phaseString[5] == 'G')
	{
		lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);
		total_queue_length += lane_queue_length[7];

		out_temp_queueTailIDs[7] = GetQueueTailVp(EAST_in_local, 4,
lane_queue_length[7], SOUTH_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
4);
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
```

```
	{
		lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
		lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
		lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);
		total_queue_length += lane_queue_length[8] + lane_queue_length[9] +
lane_queue_length[10];

		out_temp_queueTailIDs[8] = GetQueueTailVp(SOUTH_in_local, 1,
lane_queue_length[8], EAST_out_local,
NORTH_out_local);//qpg_LNK_vehicleTail(SOUTH_in_local, 1);
		out_temp_queueTailIDs[9] = GetQueueTailVp(SOUTH_in_local, 2,
lane_queue_length[9], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
2);
		out_temp_queueTailIDs[10] = GetQueueTailVp(SOUTH_in_local, 3,
lane_queue_length[10], NORTH_out_local,
NULL);//qpg_LNK_vehicleTail(SOUTH_in_local, 3);
	}
	if (phaseString[8] == 'G')
	{
		lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);
		total_queue_length += lane_queue_length[11];

		out_temp_queueTailIDs[11] = GetQueueTailVp(SOUTH_in_local, 4,
lane_queue_length[11], WEST_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
4);
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
		lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
		lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);
		total_queue_length += lane_queue_length[12] + lane_queue_length[13]
+ lane_queue_length[14];

		out_temp_queueTailIDs[12] = GetQueueTailVp(WEST_in_local, 1,
lane_queue_length[12], SOUTH_out_local,
EAST_out_local);//qpg_LNK_vehicleTail(WEST_in_local, 1);
		out_temp_queueTailIDs[13] = GetQueueTailVp(WEST_in_local, 2,
lane_queue_length[13], EAST_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
2);

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[14] = GetQueueTailVp(WEST_in_local, 3,
lane_queue_length[14], EAST_out_local, NULL);
	}
	if (phaseString[11] == 'G')
	{
		lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);
		total_queue_length += lane_queue_length[15];
```

```
            out_temp_queueTailIDs[15] = GetQueueTailVp(WEST_in_local, 4,
lane_queue_length[15], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
4);
        }

        return total_queue_length;
}

void OPTIMIZE5()
{
        //Traffic Signal Light Optimizer: Dynamic Programming Method, Max Queue
Length Optimizer: Full-Queue Discharge Version

        int i;
        int j;
        int k;
        char *currentStatus; //check if this works
    int max_queue_length = 0; //default initialization
    int BESTPHASE_INDEX = 48; //default phase is ALLRED
        int nextPhaseIndex;
        char *TESTPHASE;
        char *TESTPHASE2;
        int queue_length;
        int currentQueueIDList_out[100];
        int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8 cases, (all
green phases), NOTE: this is used in the (currentStatus == "G") branch

        int testCasesFromAllRed[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //in case
definition above causes problems, just start with constant matrix entries
        VEHICLE* temp_queueTailIDs[16];

    //add condition for iterating through the necessary test cases
    currentStatus = ALLPHASESSTATUS[currentPhaseIndex];

    if ((currentStatus == "G") && (queuesDischarged == PFALSE) &&
(currentPhaseDuration < maximumGreenDuration))
        {
         currentPhaseDuration += 1; //extend green until current phases' queues are
fully discharged

                //test section
                if (currentPhaseDuration > maxGreenReached)
                {
                        maxGreenReached = currentPhaseDuration;
                }
        }
    else if (currentStatus == "G") //CONDITION 2) described in IMA2(); check
transition to all 8 greens      //queuesDischarged == True
        {
                //NOTE: can add case later, where if max green is reached or is
within 8 seconds of being reached, then only 7 cases should be considered
                //if (currentPhaseDuration >= (maximumGreenDuration -
minimumGreenDuration))
```

```
              //int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8
cases, (all green phases), NOTE: this is declared above
        for (i = 0; i < 8; i++)
              {
            TESTPHASE = ALLPHASES[testCases[i]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(5, TESTPHASE,
&temp_queueTailIDs); //currentQueueIDList_out?

            if (queue_length > max_queue_length) //what about ties?
                    {
                max_queue_length = queue_length;
                BESTPHASE_INDEX = testCases[i]; //case index is only correct for
extending given green

                            for (k = 0; k < 16; k++)
                            {
                                    queueTailIDs[k] = temp_queueTailIDs[k]; //see if
this works
                            }
                    }
              }

        //4 possible cases
        // 1) Current green is extended 1 second (1/8)
        // 2) First movement stays green during transition to next green    (e.g.
1/5 to 1/6 with 1 continuing) (1/8)
        // 3) Second movement stays green during transition to next green (e.g.
1/5 to 2/5 with 5 continuing) (1/8)
        // 4) Both movements go to yellow before going to all red (5/8)
        if (NEXTPHASES[currentPhaseIndex][0] == BESTPHASE_INDEX) //1)
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][0]; //actually already
set above
              }
        else if (NEXTPHASES[NEXTPHASES[NEXTPHASES[currentPhaseIndex][2]][0]][0] ==
BESTPHASE_INDEX) //2) check if first movement stays green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][2];
              }
        else if (NEXTPHASES[NEXTPHASES[NEXTPHASES[currentPhaseIndex][3]][0]][0] ==
BESTPHASE_INDEX) //3) check if second movement stays green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][3];
              }
        else //4) must transition to all red on route to next green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][1];
              }

        if (BESTPHASE_INDEX == NEXTPHASES[currentPhaseIndex][0]) //check if
original green phase is extended
              {
            currentPhaseDuration += 1;
```

```
              }
        else //phase has switched to a yellow phase
              {
            nextPhaseIndex = BESTPHASE_INDEX;
                    SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]); //can
later add programatic way to find junction id for additional intersections (can be
manually specified for one intersection, as here)
            //traci.trafficlights.setRedYellowGreenState("int11",
ALLPHASES[nextPhaseIndex]) //light changes to yellow
            phase_start += currentPhaseDuration; //reset phase_start
            currentPhaseDuration = 3; //default duration for a yellow phase
            currentPhaseIndex = nextPhaseIndex;
              }
       }
    else if (currentStatus == "R") //CONDITION 1) described in IMA2()
       {
        for (j = 0; j < 8; j++)
              {
            TESTPHASE2 = ALLPHASES[testCasesFromAllRed[j]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(5, TESTPHASE2,
&temp_queueTailIDs); //, currentQueueIDList_out?
            if (queue_length > max_queue_length) //what about ties?
                    {
                max_queue_length = queue_length;
                BESTPHASE_INDEX = testCasesFromAllRed[j]; //case index is only
correct for extending given green

                            //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                            for (k = 0; k < 16; k++)
                            {
                                    queueTailIDs[k] = temp_queueTailIDs[k]; //see if
this works
                            }
                    }
              }

        //light must change to green
        nextPhaseIndex = BESTPHASE_INDEX;
        SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]); //can later add
programatic way to find junction id for additional intersections (can be manually
specified for one intersection, as here)
              //traci.trafficlights.setRedYellowGreenState("int11",
ALLPHASES[nextPhaseIndex]) //light changes to green
        phase_start += currentPhaseDuration; //reset phase_start
        currentPhaseDuration = minimumGreenDuration; //default minimum green is 8
seconds
        currentPhaseIndex = nextPhaseIndex;
              queuesDischarged = PFALSE; //check if this reset works
       }
}

void OPTIMIZE6()
```

```
{
       //Traffic Signal Light Optimizer: Dynamic Programming Method, Max Queue
Length Optimizer: Full-Queue Discharge Version, with Dual Phase Quadratic Aging
Factors

       int i;
       int j;
       int k;
       char *currentStatus; //check if this works
    int max_queue_length = 0; //default initialization
    int BESTPHASE_INDEX = 48; //default phase is ALLRED
       int nextPhaseIndex;
       char *TESTPHASE;
       char *TESTPHASE2;
       int queue_length;
       float queue_lengthXagingFactor = 0.0;
       float max_queue_lengthXagingFactor = 0.0;
       int currentQueueIDList_out[100];
       int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8 cases, (all
green phases), NOTE: this is used in the (currentStatus == "G") branch

       int testCasesFromAllRed[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //in case
definition above causes problems, just start with constant matrix entries
       VEHICLE* temp_queueTailIDs[16];

    //print('OPTIMIZE6 entered')

    //add condition for iterating through the necessary test cases
    currentStatus = ALLPHASESSTATUS[currentPhaseIndex];

    if ((currentStatus == "G") && (queuesDischarged == PFALSE) &&
(currentPhaseDuration < maximumGreenDuration))
       {
        currentPhaseDuration += 1; //extend green until current phases' queues are
fully discharged

              //test section
              if (currentPhaseDuration > maxGreenReached)
              {
                     maxGreenReached = currentPhaseDuration;
              }
       }
    else if (currentStatus == "G") //CONDITION 2) described in IMA2(); check
transition to all 8 greens      //queuesDischarged == True
       {
              //NOTE: can add case later, where if max green is reached or is
within 8 seconds of being reached, then only 7 cases should be considered
              //if (currentPhaseDuration >= (maximumGreenDuration -
minimumGreenDuration))

              //int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8
cases, (all green phases), NOTE: this is declared above
        for (i = 0; i < 8; i++)
              {
```

```
            TESTPHASE = ALLPHASES[testCases[i]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(5, TESTPHASE,
&temp_queueTailIDs); //currentQueueIDList_out?
                     queue_lengthXagingFactor = queue_length * agingFactor[i];

            if (queue_lengthXagingFactor > max_queue_lengthXagingFactor) //what
about ties?
                     {
                max_queue_lengthXagingFactor = queue_lengthXagingFactor;
                BESTPHASE_INDEX = testCases[i]; //case index is only correct for
extending given green

                             //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                             for (k = 0; k < 16; k++)
                             {
                                    queueTailIDs[k] = temp_queueTailIDs[k]; //see if
this works
                             }
                     }
              }

        //4 possible cases
        // 1) Current green is extended 1 second (1/8)
        // 2) First movement stays green during transition to next green    (e.g.
1/5 to 1/6 with 1 continuing) (1/8)
        // 3) Second movement stays green during transition to next green (e.g.
1/5 to 2/5 with 5 continuing) (1/8)
        // 4) Both movements go to yellow before going to all red (5/8)
        if (NEXTPHASES[currentPhaseIndex][0] == BESTPHASE_INDEX) //1)
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][0]; //actually already
set above
              }
        else if (NEXTPHASES[NEXTPHASES[NEXTPHASES[currentPhaseIndex][2]][0]][0] ==
BESTPHASE_INDEX) //2) check if first movement stays green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][2];
              }
        else if (NEXTPHASES[NEXTPHASES[NEXTPHASES[currentPhaseIndex][3]][0]][0] ==
BESTPHASE_INDEX) //3) check if second movement stays green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][3];
              }
        else //4) must transition to all red on route to next green
              {
            BESTPHASE_INDEX = NEXTPHASES[currentPhaseIndex][1];
              }

        if (BESTPHASE_INDEX == NEXTPHASES[currentPhaseIndex][0]) //check if
original green phase is extended
              {
            currentPhaseDuration += 1;
```

```
              }
        else //phase has switched to a yellow phase
              {
            nextPhaseIndex = BESTPHASE_INDEX;
                    SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]); //can
later add programmatic way to find junction id for additional intersections (can
be manually specified for one intersection, as here)
            //light changes to yellow
            phase_start += currentPhaseDuration; //reset phase_start
            currentPhaseDuration = 3; //default duration for a yellow phase
            currentPhaseIndex = nextPhaseIndex;
              }
       }
    else if (currentStatus == "R") //CONDITION 1) described in IMA2()
       {
        for (j = 0; j < 8; j++)
              {
            TESTPHASE2 = ALLPHASES[testCasesFromAllRed[j]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(5, TESTPHASE2,
&temp_queueTailIDs); //, currentQueueIDList_out?
                    queue_lengthXagingFactor = queue_length * agingFactor[j];

                    if (queue_lengthXagingFactor > max_queue_lengthXagingFactor)
//what about ties?
                    {
                max_queue_lengthXagingFactor = queue_lengthXagingFactor;
                BESTPHASE_INDEX = testCasesFromAllRed[j]; //case index is only
correct for extending given green

                           //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                           for (k = 0; k < 16; k++)
                           {
                                  queueTailIDs[k] = temp_queueTailIDs[k]; //see if
this works
                           }

                                  }
              }

        //light must change to green
        nextPhaseIndex = BESTPHASE_INDEX;
        SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]); //can later add
programmatic way to find junction id for additional intersections (can be manually
specified for one intersection, as here)
              //light changes to green
        phase_start += currentPhaseDuration; //reset phase_start
        currentPhaseDuration = minimumGreenDuration; //default minimum green is 8
seconds
        currentPhaseIndex = nextPhaseIndex;
              queuesDischarged = PFALSE; //check if this reset works
       }
}
```

```
void IMA2()
{
       //Intersection Management Agent queries Optimize5() if necessary
    //Optimization is only necessary under the following 2 conditions
    //CONDITION 1) An ALLRED phase has just expired, OPTIMIZE#() must decide which
green phase to switch to
    //CONDITION 2) A double green phase is past its min green time of 8 seconds

       char *currentStatus; //check if this works
       float currentTime;
       int nextPhaseIndex;

    // determine if OPTIMIZE# needs to be called
       currentTime = qpg_CFG_simulationTime(); //time returned by paramics should
have units of seconds

    //check light status as defined in ALLPHASESSTATUS
    currentStatus = ALLPHASESSTATUS[currentPhaseIndex];
    if (currentStatus == "Y") //light remains yellow until changed
       {
        if ((currentTime - phase_start) >= currentPhaseDuration)
//currentPhaseDuration should be 3 seconds here
              {
            //change light to appropriate red
            nextPhaseIndex = NEXTPHASES[currentPhaseIndex][0];
                     SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]); //can
later add programmatic way to find junction id for additional intersections (can
be manually specified for one intersection, as here)
            //light changes to red
            phase_start += currentPhaseDuration; //reset phase_start
            currentPhaseDuration = 1; //all phase combinations with status == "R"
currently have a fixed duration of 1 second
            currentPhaseIndex = nextPhaseIndex;
              }
       }
    else if (currentStatus == "R") //light remains red until changed
       {
        if ((currentTime - phase_start) >= currentPhaseDuration)
//currentPhaseDuration should be 1 second here
              {
            //check if red is all red
            if (currentPhaseIndex == 48) //48 is the index for ALLRED
                     {
                OPTIMIZE6(); //5
                     }
            else //must be a single green phase, therefore, transition to a double
green phase (see FlexibleTrafficLightStateMachine_colored.vsd)
                     {
                //change light to appropriate green
                nextPhaseIndex = NEXTPHASES[currentPhaseIndex][0];
                            SetRedYellowGreenState(5, ALLPHASES[nextPhaseIndex]);
//can later add programatic way to find junction id for additional intersections
(can be manually specified for one intersection, as here)
```

```
//light changes to green
                phase_start += currentPhaseDuration; //reset phase_start
                currentPhaseDuration = minimumGreenDuration; //reset current phase
duration to min green
                currentPhaseIndex = nextPhaseIndex;
                            queuesDischarged = PFALSE; //check if this reset works
                     }
              }
       }
       else if (currentStatus == "G") //light remains green until changed, should
remain green for at least the duration of min green
       {
        if ((currentTime - phase_start) >= currentPhaseDuration) //should be at
least 8 seconds, OPTIMIZE//() can extend the duration in increments of 1 second
              {
            OPTIMIZE6();    //5
              }
       }
}

Bool QueuesDischargedCheck(int intersectionID, char *phaseString)
{
       Bool temp_queuesDischarged = PTRUE; //initialization must be 'true'

       //declare pointers (these are the links connected to the junction specified
by the intersectionID parameter)
       LINK* WEST_in_local; //check if this works
       LINK* NORTH_in_local;
       LINK* EAST_in_local;
       LINK* SOUTH_in_local;

       //outbound links are not needed
       /*LINK* WEST_out_local;
       LINK* NORTH_out_local;
       LINK* EAST_out_local;
       LINK* SOUTH_out_local;*/

       //NOTE: can use intersectionID later to add additional intersections
       //NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

       //METHOD 1: manual specification of links connected to junction //NOTE: can
later add a function to programatically find these links (in the case of multiple
junctions)
       WEST_in_local = qpg_NET_link("11:5"); //this set of link pointers will only
work for the isolated intersection network based on
"Isolated_CustomAdaptive_Four_way_Intersection_test1"
       NORTH_in_local = qpg_NET_link("13:5");
       EAST_in_local = qpg_NET_link("10:5");
       SOUTH_in_local = qpg_NET_link("12:5");

       //NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
```

```
	//NOTE: Queue is discharged when the front bumper of the designated tail
vehicle crosses the stop bar
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(queueTailIDs[0]) == NORTH_in_local) ||
(qpg_VHC_link(queueTailIDs[1]) == NORTH_in_local) ||
(qpg_VHC_link(queueTailIDs[2]) == NORTH_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[2] == 'G')
	{
		if (qpg_VHC_link(queueTailIDs[3]) == NORTH_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(queueTailIDs[4]) == EAST_in_local) ||
(qpg_VHC_link(queueTailIDs[5]) == EAST_in_local) || (qpg_VHC_link(queueTailIDs[6])
== EAST_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[5] == 'G')
	{
		if (qpg_VHC_link(queueTailIDs[7]) == EAST_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(queueTailIDs[8]) == SOUTH_in_local) ||
(qpg_VHC_link(queueTailIDs[9]) == SOUTH_in_local) ||
(qpg_VHC_link(queueTailIDs[10]) == SOUTH_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[8] == 'G')
	{
		if (qpg_VHC_link(queueTailIDs[11]) == SOUTH_in_local)
```

```
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(queueTailIDs[12]) == WEST_in_local) ||
(qpg_VHC_link(queueTailIDs[13]) == WEST_in_local) ||
(qpg_VHC_link(queueTailIDs[14]) == WEST_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[11] == 'G')
	{
		if (qpg_VHC_link(queueTailIDs[15]) == WEST_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	return temp_queuesDischarged;
}

void UpdateTimeLastServed(float currentTime_in)
{
	//char *lastPhaseString = ALLPHASES[lastPhaseIndex]; //check if this works
	char *currentPhaseString = ALLPHASES[currentPhaseIndex];
	int i = 0;

	for (i = 0; i < strlen(currentPhaseString); i++) //length should be 12
	{
		//if ((lastPhaseString[i] == 'G') && (currentPhaseString[i] == 'y'))
//update "time last served" when a green transitions to yellow
		if (currentPhaseString[i] == 'G') //update timeLastServed[] while
the light is green
		{
			//Description: The strings use the following characters: 1) G
= green, 2) y = yellow, 3) R = Right Turn On Red, 4) r = red
			//		The movements are ordered cw, starting from the
North_in right turn movement, or more explicitly the 12 movements are as follows
(NEMA phase equivalent in parantheses):
			//		1. North_in, West_out  (4)
			//		2. North_in, South_out (4)
			//		3. North_in, East_out  (7)
			//		4. East_in, North_out  (6)
			//		5. East_in, West_out   (6)
			//		6. East_in, South_out  (1)
			//		7. South_in, East_out  (8)
			//		8. South_in, North_out (8)
			//		9. South_in, West_out  (3)
```

```
                        //                10. West_in, South_out  (2)
                        //                11. West_in, East_out   (2)
                        //                12. West_in, North_out  (5)

                        if ((i == 0) || (i == 1))
                        {
                                timeLastServed[3] = currentTime_in; //NEMA phase 4
                        }
                        else if (i == 2)
                        {
                                timeLastServed[6] = currentTime_in; //NEMA phase 7
                        }
                        else if ((i == 3) || (i == 4))
                        {
                                timeLastServed[5] = currentTime_in; //NEMA phase 6
                        }
                        else if (i == 5)
                        {
                                timeLastServed[0] = currentTime_in; //NEMA phase 1
                        }
                        else if ((i == 6) || (i == 7))
                        {
                                timeLastServed[7] = currentTime_in; //NEMA phase 8
                        }
                        else if (i == 8)
                        {
                                timeLastServed[2] = currentTime_in; //NEMA phase 3
                        }
                        else if ((i == 9) || (i == 10))
                        {
                                timeLastServed[1] = currentTime_in; //NEMA phase 2
                        }
                        else if (i == 11)
                        {
                                timeLastServed[4] = currentTime_in; //NEMA phase 5
                        }
                }
        }

}

void UpdateAgingTime(float currentTime_in)
{
        int i = 0;

        for (i = 0; i < 8; i++) //8 is number of individual phase movements
        {
                agingTime[i] = currentTime_in - timeLastServed[i];
        }
}

float CalculateAgingFactor(float dualPhaseAgingTime)
{
```

```
	//aging factor equation is a quadratic fitting three points perfectly (0,
1), (30, 2), and (120, 10)
	//This means that an input of 120 seconds time leads to an aging factor of
10, (the queue lengths are multiplied by a factor of 10)
	return 0.000462963 * dualPhaseAgingTime * dualPhaseAgingTime + 0.0194444 *
dualPhaseAgingTime + 1;
}

void UpdateAgingFactors()
{
	//aging factor multiplies queue length to modulate fairness, based on
elapsed time since a movement has been served
	//NOTE: aging factors refers to dual phases: 1/5, 1/6, 2/5, 2/6, 3/7, 3/8,
4/7, 4/8
	agingFactor[0] = CalculateAgingFactor((agingTime[0] + agingTime[4])/2);
//NEMA dual phase 1/5
	agingFactor[1] = CalculateAgingFactor((agingTime[0] + agingTime[5])/2);
//NEMA dual phase 1/6
	agingFactor[2] = CalculateAgingFactor((agingTime[1] + agingTime[4])/2);
//NEMA dual phase 2/5
	agingFactor[3] = CalculateAgingFactor((agingTime[1] + agingTime[5])/2);
//NEMA dual phase 2/6
	agingFactor[4] = CalculateAgingFactor((agingTime[2] + agingTime[6])/2);
//NEMA dual phase 3/7
	agingFactor[5] = CalculateAgingFactor((agingTime[2] + agingTime[7])/2);
//NEMA dual phase 3/8
	agingFactor[6] = CalculateAgingFactor((agingTime[3] + agingTime[6])/2);
//NEMA dual phase 4/7
	agingFactor[7] = CalculateAgingFactor((agingTime[3] + agingTime[7])/2);
//NEMA dual phase 4/8
}

void UpdateAgingFactors2()
{
	//UpdateAgingFactor2 accounts for the number of vehicles per individual
movement
	//aging factor multiplies queue length to modulate fairness, based on
elapsed time since a movement has been served
	//NOTE: aging factors refers to dual phases: 1/5, 1/6, 2/5, 2/6, 3/7, 3/8,
4/7, 4/8
	int individual_phase_queue_lengths[8] = {1};
	int *p;
	int i;
	// 1) First get queue lengths for each individual NEMA phase
	p = Find_All_Individual_Phase_Queue_Lengths(5); //can update later to
account for multiple intersections, ...check if this works

	for (i = 0; i < 8; i++)
	{
		individual_phase_queue_lengths[i] = *(p + i);
	}
```

```
	/*qps_GUI_printf("phase 1 queue length: %d \n",
individual_phase_queue_lengths[0]);
	qps_GUI_printf("phase 2 queue length: %d \n",
individual_phase_queue_lengths[1]);
	qps_GUI_printf("phase 3 queue length: %d \n",
individual_phase_queue_lengths[2]);
	qps_GUI_printf("phase 4 queue length: %d \n",
individual_phase_queue_lengths[3]);
	qps_GUI_printf("phase 5 queue length: %d \n",
individual_phase_queue_lengths[4]);
	qps_GUI_printf("phase 6 queue length: %d \n",
individual_phase_queue_lengths[5]);
	qps_GUI_printf("phase 7 queue length: %d \n",
individual_phase_queue_lengths[6]);
	qps_GUI_printf("phase 8 queue length: %d \n",
individual_phase_queue_lengths[7]);*/

	//NOTE: the max function is used in the denominator of the calculations
below to avoid dividing by 0 when there are no vehicles on either phase
	agingFactor[0] = CalculateAgingFactor(((agingTime[0] *
individual_phase_queue_lengths[0]) + (agingTime[4] *
individual_phase_queue_lengths[4]))/(MAX(individual_phase_queue_lengths[0] +
individual_phase_queue_lengths[4], 1))); //NEMA dual phase 1/5
	agingFactor[1] = CalculateAgingFactor(((agingTime[0] *
individual_phase_queue_lengths[0]) + (agingTime[5] *
individual_phase_queue_lengths[5]))/(MAX(individual_phase_queue_lengths[0] +
individual_phase_queue_lengths[5], 1))); //NEMA dual phase 1/6
	agingFactor[2] = CalculateAgingFactor(((agingTime[1] *
individual_phase_queue_lengths[1]) + (agingTime[4] *
individual_phase_queue_lengths[4]))/(MAX(individual_phase_queue_lengths[1] +
individual_phase_queue_lengths[4], 1))); //NEMA dual phase 2/5
	agingFactor[3] = CalculateAgingFactor(((agingTime[1] *
individual_phase_queue_lengths[1]) + (agingTime[5] *
individual_phase_queue_lengths[5]))/(MAX(individual_phase_queue_lengths[1] +
individual_phase_queue_lengths[5], 1))); //NEMA dual phase 2/6
	agingFactor[4] = CalculateAgingFactor(((agingTime[2] *
individual_phase_queue_lengths[2]) + (agingTime[6] *
individual_phase_queue_lengths[6]))/(MAX(individual_phase_queue_lengths[2] +
individual_phase_queue_lengths[6], 1))); //NEMA dual phase 3/7
	agingFactor[5] = CalculateAgingFactor(((agingTime[2] *
individual_phase_queue_lengths[2]) + (agingTime[7] *
individual_phase_queue_lengths[7]))/(MAX(individual_phase_queue_lengths[2] +
individual_phase_queue_lengths[7], 1))); //NEMA dual phase 3/8
	agingFactor[6] = CalculateAgingFactor(((agingTime[3] *
individual_phase_queue_lengths[3]) + (agingTime[6] *
individual_phase_queue_lengths[6]))/(MAX(individual_phase_queue_lengths[3] +
individual_phase_queue_lengths[6], 1))); //NEMA dual phase 4/7
	agingFactor[7] = CalculateAgingFactor(((agingTime[3] *
individual_phase_queue_lengths[3]) + (agingTime[7] *
individual_phase_queue_lengths[7]))/(MAX(individual_phase_queue_lengths[3] +
individual_phase_queue_lengths[7], 1))); //NEMA dual phase 4/8
}

void qpx_CLK_startOfSimLoop()
```

```
{
	float currentTime = qpg_CFG_simulationTime(); //elapsed simulation time in
seconds

	if ((currentTime > 3*DELTA) && (currentTime < 5*DELTA)) //set to correct
lights after the first time step
	{
		qps_GUI_printf("initial phase entered at time %f \n", currentTime);
		//set initial priorities, 1/5 green
		qps_LNK_priority(NORTH_in, WEST_out, APIPRI_MINOR);  //4
		qps_LNK_priority(NORTH_in, SOUTH_out, APIPRI_BARRED); //4
		qps_LNK_priority(NORTH_in, EAST_out, APIPRI_BARRED);  //7

		qps_LNK_priority(EAST_in, NORTH_out, APIPRI_BARRED);  //6
		qps_LNK_priority(EAST_in, WEST_out, APIPRI_BARRED);   //6
		qps_LNK_priority(EAST_in, SOUTH_out, APIPRI_MAJOR);  //1

		qps_LNK_priority(SOUTH_in, EAST_out, APIPRI_MINOR);  //8
		qps_LNK_priority(SOUTH_in, NORTH_out, APIPRI_BARRED); //8
		qps_LNK_priority(SOUTH_in, WEST_out, APIPRI_BARRED);  //3

		qps_LNK_priority(WEST_in, SOUTH_out, APIPRI_BARRED);  //2
		qps_LNK_priority(WEST_in, EAST_out, APIPRI_BARRED);  //2
		qps_LNK_priority(WEST_in, NORTH_out, APIPRI_MAJOR); //5
	}

	if (fabs(currentTime - (int)currentTime) < DELTA) //update aging time and
aging factor, prior to calling the IMA
	{
		//sequence of these functions is important 1)TimeLastServed, 2)
AgingTime, 3) AgingFactor
		UpdateTimeLastServed(currentTime);
		UpdateAgingTime(currentTime);
		// UpdateAgingFactors(); // replaced with call to
UpdateAgingFactor2()
		UpdateAgingFactors2();
	}

		IMA2(); //start calling immediately, initial phase is 1/5, check if

	if (fabs(currentTime - (int)currentTime) < DELTA) //update queuesDischarged
once a second
	{
		//check queueTailIDs array to update queuesDischarged boolean
		if (currentTime > (minimumGreenDuration + 4)) //prevent check from
occurring during the first phase of the simulation (as there are no queues to
check at the beginning of the simulation), NOTE: need to add logic to accommodate
larger networks with additional intersections
		{
			queuesDischarged = QueuesDischargedCheck(5,
ALLPHASES[currentPhaseIndex]); //what about for the first phase of the simulation?
		}
	}
```

```
	//update lastPhaseIndex
	lastPhaseIndex = currentPhaseIndex; //since this is at the beginning of the
time step, currentPhaseIndex actually refers to the resulting calculations of the
previous time step

	//qps_GUI_printf("qpx_CLK_startOfSimLoop() entered at time %f , DELTA = %f
\n", currentTime, DELTA);
}

float in_qpo_CFM_Speed(LINK* link, VEHICLE* Vp, float CFM_Speed)
{
	float v_lmt = qpg_LNK_speedlimit(link)/2.2369 + 10;
	float currTime = qpg_CFG_simulationTime();
	float calSpeed, recSpeed, currVel = qpg_VHC_speed(Vp); //calSpeed is used
for ESH
	int linkid = qpg_LNK_index(link), mode, currZone = qpg_LNK_zone(link);
	float timeStep = qpg_CFG_timeStep();

	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp); // members of
current vehicle (pointer is from input parameter)

	float acceleration = 0; // used for acceleration color scheme
	float colorScale = 0;


	// calculate emissions
	if (fabs(currTime - (int)(currTime+0.5)) > DELTA || currZone == myVeh-
>origin || currZone == myVeh->dest) //these are conditions under which emissions
should not be calculated
		goto CACC;
	if (myVeh->first2sec < 3)
	{
		myVeh->tm[myVeh->first2sec] = currTime;
		myVeh->linkID[myVeh->first2sec] = linkid;
		myVeh->vel[myVeh->first2sec] = currVel;
		myVeh->first2sec++;
		goto CACC;
	}
	calEmissions(myVeh, currTime, currVel, linkid); //emissions are calculated
every three seconds

CACC:


	return CFM_Speed;

}

float qpo_CFM_leadSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
	float CFM_Speed = qpg_CFM_leadSpeed(link, Vp, ahead);
	return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
}
```

```
float qpo_CFM_followSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
	float CFM_Speed = qpg_CFM_followSpeed(link, Vp, ahead);
	return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
}

void qpx_VHC_arrive(VEHICLE* Vp, LINK* link, ZONE* zone)
{
	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp);
	TT_veh[counter_arrived] = qpg_VHC_existTime(Vp);
	VMT += myVeh->total_dist; //myVeh->tripDist; //tripDist was originally used
here (when RTR was working) ...try total_dist, check if it is the same as tripDist
with RTR (if RTR is not bugged)
	VHT += TT_veh[counter_arrived];

	counter_arrived++;
	free(myVeh);
}

void  qpx_NET_complete()
{
	FILE *fsbs, *fsum, *fopMode, *fVSP, *fTT, *fsbs_e, *fsum_e, *fopMode_e,
*fVSP_e, *fTT_e, *fsbs_n, *fsum_n, *fopMode_n, *fVSP_n, *fTT_n;
	int i;

	VMT = VMT*meter2mile;
	dist_sum = dist_sum*meter2mile;
	energy_sum = energy_sum/dist_sum/3600;
	CO2_sum = CO2_sum/dist_sum/3600;
	CO_sum = CO_sum/dist_sum/3600;
	HC_sum = HC_sum/dist_sum/3600;
	NOx_sum = NOx_sum/dist_sum/3600;
	PM_sum = PM_sum/dist_sum/3600;

	fsum = fopen(out2Path, "w");
	fsum_e = fopen(out21Path, "w");
	fsum_n = fopen(out22Path, "w");

	if (fsum == NULL || fsum_e == NULL || fsum_n == NULL) //never actually
executes
	{
		qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);

		qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
		qps_GUI_printf("Couldn't open file. (simulation ended, results not
logged)");
	}

	fprintf(fsum,
"Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi),VHT(sec/veh),VMT(
mi/veh)\n");
```

```
	if (counter_arrived > 0)
	{
		fprintf(fsum, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);
		fprintf(fsum, "%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
	}

	fclose(fsum);
	fclose(fsum_e);
	fclose(fsum_n);

	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum, CO2_sum,
CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived, VMT/counter_arrived);

	qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
}
```

Appendix D: Corridor-Level Connected Vehicle Signal Optimization Code

The following code is C/Paramics plugin code for corridor-level connected vehicle signal optimization.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "programmer.h"
#include <windows.h>
#include <assert.h>

#define MIN(a,b)        ((a) < (b) ? (a) : (b))
#define MAX(a,b)        ((a) > (b) ? (a) : (b))
#define ARRAYLENGTH(a)  (sizeof(a) / sizeof((a)[0])) //number of elements in an
array
#define NDEBUG //comment out this line to activate assert statements

typedef struct veh_profile VEHICLE_DATA;
struct veh_profile
{
	float A; // rolling coefficient in kW*sec/meter
	float B; // rotation coefficient in kW*sec^2/meter^2
	float C; // drag coefficient in kW*sec^3/meter^3
	float M; // vehicle source mass in metric tons
	float f; // fixed mass factor in metric tons
	int sourceType; // vehicle source type
	int ID; // vehicle ID
	float theta; // road grade angle
	float vel[4]; // 3-second velocities
	float acc[4]; // 3-second accelerations
	int linkID[4]; // 3-second link IDs
	float VSP[4]; // 3-second VSPs
	int mode[4]; // 3-second modes
	float tm[4]; // 3-second time stamps
	int first2sec; // the first 2 seconds vehicle enters the network
	float energy[4], CO2[4], CO[4], HC[4], NOx[4], PM[4]; // 3-second emission
data
	int origin; // vehicle's origin zone
	int dest; // vehicle's destination zone
	float x_destNode; // x coordinate of start node of vehicle's destination
zone
	float y_destNode; // y coordinate of start node of vehicle's destination
zone
	float tripDist; // vehicle's total trip distance
```

```
	int bound; // 1: northbound; 2: southbound: 0 otherwise
	LINK* currentLink; //pointer to most recent current link
	float prevSpeed; //used to calculate acceleration
	float total_energy; //trip energy for ego-vehicle
	float total_dist; //trip distance for ego-vehicle (may replace tripDist)

	//CV Adaptive Signal Control Variables
	//NOTE: current network delay avoids having to reset current delay between
intersections (optimize for network as a whole, instead of optimizing each
intersection independently, ...can test independent approach later)
	float currentNetworkDelay; //current network delay includes delay due to
idling and other delays resulting in the vehicle being under the speed limit
};

//Parameters (for Adaptive/Optimal Signal Control)

//Adaptive/Optimal Signal Control Variables
NODE* n[1]; //change this number to total number of signals within the network

//Independent NEMA phases (2 through-lane version)
//NOTE: These strings differ from the SUMO strings, due to the fact that Paramics
does not have independent lane control like SUMO does
//	Therefore, the characters in the SUMO strings refer to lanes, whereas the
characters in the Paramics strings below refer to movements (total of 12
movements)
//	One beneficial byproduct is that the Paramics strings remain the same for
all 4-way intersections regardless of the number of lanes, or if certain lanes
share movements.
//Description: The strings use the following characters: 1) G = green, 2) y =
yellow, 3) R = Right Turn On Red, 4) r = red
//		The movements are ordered cw, starting from the North_in right turn
movement, or more explicitly the 12 movements are as follows (NEMA phase
equivalent in parantheses):
//		1. North_in, West_out  (4)
//		2. North_in, South_out (4)
//		3. North_in, East_out  (7)
//		4. East_in, North_out  (6)
//		5. East_in, West_out   (6)
//		6. East_in, South_out  (1)
//		7. South_in, East_out  (8)
//		8. South_in, North_out (8)
//		9. South_in, West_out  (3)
//		10. West_in, South_out  (2)
//		11. West_in, East_out   (2)
//		12. West_in, North_out  (5)
//// without RTOR logic:
//const char *ONEGREEN = "rrrrrGrrrrrr";//"rrrrrrrGrrrrrrrr";
//const char *ONEYELLOW = "rrrrryrrrrrr";//"rrrrrrryrrrrrrrr";
//const char *TWOGREEN = "rrrrrrrrrGGr";//"rrrrrrrrrrrrGGGr";
//const char *TWOYELLOW = "rrrrrrrrryyr";//"rrrrrrrrrrrryyyr";
//const char *THREEGREEN = "rrrrrrrrGrrr";//"rrrrrrrrrrrGrrrr";
//const char *THREEYELLOW = "rrrrrrrryrrr";//"rrrrrrrrrrryrrrr";
//const char *FOURGREEN = "GGrrrrrrrrrr";//"GGGrrrrrrrrrrrrr";
//const char *FOURYELLOW = "yyrrrrrrrrrr";//"yyyrrrrrrrrrrrrr";
```

```
//const char *FIVEGREEN = "rrrrrrrrrrrG";//"rrrrrrrrrrrrrrrG";
//const char *FIVEYELLOW = "rrrrrrrrrrry";//"rrrrrrrrrrrrrrry";
//const char *SIXGREEN = "rrrGGrrrrrrr";//"rrrrGGGrrrrrrrrr";
//const char *SIXYELLOW = "rrryyrrrrrrr";//"rrrryyyrrrrrrrrr";
//const char *SEVENGREEN = "rrGrrrrrrrrr";//"rrrGrrrrrrrrrrrr";
//const char *SEVENYELLOW = "rryrrrrrrrrr";//"rrryrrrrrrrrrrrr";
//const char *EIGHTGREEN = "rrrrrrGGrrrr";//"rrrrrrrrGGGrrrrr";
//const char *EIGHTYELLOW = "rrrrrryyrrrr";//"rrrrrrrryyyrrrrr";
//
//const char *ALLRED = "rrrrrrrrrrrr";//"rrrrrrrrrrrrrrrr";

// with RTOR logic: combination rule: G > y > r > R
const char *ONEGREEN = "RrrRrGRrrrrr";//"rrrrrrrGrrrrrrrr";
const char *ONEYELLOW = "RrrRryRrrrrr";//"rrrrrrryrrrrrrrr";
const char *TWOGREEN = "RrrRrrRrrGGr";//"rrrrrrrrrrrrGGGr";
const char *TWOYELLOW = "RrrRrrRrryyr";//"rrrrrrrrrrrryyyr";
const char *THREEGREEN = "rrrRrrRrGRrr";//"rrrrrrrrrrrGrrrr";
const char *THREEYELLOW = "rrrRrrRryRrr";//"rrrrrrrrrrryrrrr";
const char *FOURGREEN = "GGrRrrRrrRrr";//"GGGrrrrrrrrrrrrr";
const char *FOURYELLOW = "yyrRrrRrrRrr";//"yyyrrrrrrrrrrrrr";
const char *FIVEGREEN = "RrrrrrRrrRrG";//"rrrrrrrrrrrrrrrG";
const char *FIVEYELLOW = "RrrrrrRrrRry";//"rrrrrrrrrrrrrrry";
const char *SIXGREEN = "RrrGGrRrrRrr";//"rrrrGGGrrrrrrrrr";
const char *SIXYELLOW = "RrryyrRrrRrr";//"rrrryyyrrrrrrrrr";
const char *SEVENGREEN = "RrGRrrrrrRrr";//"rrrGrrrrrrrrrrrr";
const char *SEVENYELLOW = "RryRrrrrrRrr";//"rrryrrrrrrrrrrrr";
const char *EIGHTGREEN = "RrrRrrGGrRrr";//"rrrrrrrrGGGrrrrr";
const char *EIGHTYELLOW = "RrrRrryyrRrr";//"rrrrrrrryyyrrrrr";

const char *ALLRED = "rrrrrrrrrrrr";//"rrrrrrrrrrrrrrrr";

//Constants
const float GRAV = 9.8; // gravity coefficient in meter/second^2
const float INDEX = 2.23693629; // conversion index from m/s to mph
const float mps2mph = 2.23693629; //same as INDEX (consolidate this later)
const float meter2mile = 0.000621371;
const float vehicleLength = 3.9990; //4.8768 was for SUMO; // can replact this by
using the qpg_VHC_length(VEHICLE* vehicle) function (needed for multiple vehicle
types)
float DELTA = 0.1; // default small value, later modified in qpx_NET_postOpen()
// for MOVES
float moves[63][41][6]; // moves[regClass][opMode][emissionCategory]
int modeBins[50] = {0}; // opMode distribution

// overall output data
char out1Path[150]; // file "data_sbs.dat"
char out2Path[150]; // file "MOVES_sum.dat"
char out3Path[150]; // file "opMode.dat"
char out4Path[150]; // file "VSP.dat"
char out5Path[150]; // file "TT.dat"
// for equipped vehicle only
char out11Path[150]; // file "data_sbs_e.dat"
char out21Path[150]; // file "MOVES_sum_e.dat"
char out31Path[150]; // file "opMode_e.dat"
```

```
char out41Path[150]; // file "VSP_e.dat"
char out51Path[150]; // file "TT_e.dat"
// for unequipped vehicle only
char out12Path[150]; // file "data_sbs_n.dat"
char out22Path[150]; // file "MOVES_sum_n.dat"
char out32Path[150]; // file "opMode_n.dat"
char out42Path[150]; // file "VSP_n.dat"
char out52Path[150]; // file "TT_n.dat"

float TT_veh[500000]; // travel time of each vehicle

int counter_released = 0;
int counter_released_c = 0; //vehicles released that will travel through the
entire main corridor
int counter_released_nc = 0; //vehicles released that will not travel through the
entire main corridor

int counter_arrived = 0;
int counter_arrived_c = 0; //vehicles arriving after traveling through the entire
main corridor
int counter_arrived_nc = 0; //vehicles arriving, not having traveled through the
entire main corridor

// aggregated data
// define variables for overall traffic
double VMT = 0, VHT = 0, dist_sum = 0, energy_sum = 0, CO2_sum = 0, CO_sum = 0,
HC_sum = 0, NOx_sum = 0, PM_sum = 0;
// define variables for coordinated phase vehicles only (mainstream vehicles)
double VMT_c = 0, VHT_c = 0, dist_sum_c = 0, energy_sum_c = 0, CO2_sum_c = 0,
CO_sum_c = 0, HC_sum_c = 0, NOx_sum_c = 0, PM_sum_c = 0;
// define variables for non-coordinated phase vehicles only (non-mainstream
vehicles)
double VMT_nc = 0, VHT_nc = 0, dist_sum_nc = 0, energy_sum_nc = 0, CO2_sum_nc = 0,
CO_sum_nc = 0, HC_sum_nc = 0, NOx_sum_nc = 0, PM_sum_nc = 0;

//Acceleration/Deceleration Constraints (ECO-CACC)
float max_accel_mpss = 3.5; //3.5 m/s^2 (NOTE: in this case, the selected values
were set to match the selected profile (overall max), other values may be set
(based on speed, or platoon member comfort))
float min_decel_mpss = -7.5; //-7.5 m/s^2

//DEBUG global variables
int DEBUG_FLAG = 0;

//Drawing variables
static float llx = 0.0f, lly = 0.0f, urx = 0.0f, ury = 0.0f;
static Bool rhd;

static void translatePoint(float *x, float *y)
{
	*x = rhd ? *x + llx : *x - llx;
	*y = *y - lly;
}
```

```
char *CombinePhases(const char *phase1, const char *phase2)
{
	//Combine two "single NEMA phase" phase strings into one phase string with
the rule G > y > r > R
	int i;
	static char x[13] = "rrrrrrrrrrrr"; //NOTE: remember to use static local
variables when dealing with local pointers being passed out of a function!
	char *output;

	//qps_GUI_printf("%s %d \n", x, strlen(x));

	//int len = strlen(phase1);
	for (i = 0; i < strlen(phase1); i++)
	{
		if ((phase1[i] == 'G') || (phase2[i] == 'G'))
			x[i] = 'G';
		else if ((phase1[i] == 'y') || (phase2[i] == 'y'))
			x[i] = 'y';
		else if ((phase1[i] == 'r') || (phase2[i] == 'r'))
			x[i] = 'r';
		else //must be "R"
			x[i] = 'R';
	}

	output = x;
	//qps_GUI_printf("%s %d \n", output, strlen(output)); //x is 2x-1 length
instead of x length when the definition x[x length] is used, and x is not
converting back to a char * properly

	return output;
}

const char *ALLPHASES[49] = {"RrrrrGRrrrrG",
                                                "RrrrryRrrrry",
                                                "RrrrrGRrrrry",
                                                "RrrRrGRrrrrr",
                                                "RrrrryRrrrrG",
                                                "RrrrrrRrrRrG",
                                                "RrrGGGRrrrrr",
                                                "RrryyyRrrrrr",
                                                "RrryyGRrrrrr",
                                                "RrrRrGRrrrrr",
                                                "RrrGGyRrrrrr",
                                                "RrrGGrRrrRrr",
                                                "RrrrrrRrrGGG",
                                                "RrrrrrRrryyy",
                                                "RrrrrrRrrGGy",
                                                "RrrRrrRrrGGr",
                                                "RrrrrrRrryyG",
                                                "RrrrrrRrrRrG",
                                                "RrrGGrRrrGGr",
                                                "RrryyrRrryyr",
                                                "RrryyrRrrGGr",
                                                "RrrRrrRrrGGr",
```

```
                                                        "RrrGGrRrryyr",
                                                        "RrrGGrRrrRrr",
                                                        "rrGRrrrrGRrr",
                                                        "rryRrrrryRrr",
                                                        "rryRrrrrGRrr",
                                                        "rrrRrrRrGRrr",
                                                        "rrGRrrrryRrr",
                                                        "RrGRrrrrrRrr",
                                                        "rrrRrrGGGRrr",
                                                        "rrrRrryyyRrr",
                                                        "rrrRrryyGRrr",
                                                        "rrrRrrRrGRrr",
                                                        "rrrRrrGGyRrr",
                                                        "RrrRrrGGrRrr",
                                                        "GGGRrrrrrRrr",
                                                        "yyyRrrrrrRrr",
                                                        "GGyRrrrrrRrr",
                                                        "GGrRrrRrrRrr",
                                                        "yyGRrrrrrRrr",
                                                        "RrGRrrrrrRrr",
                                                        "GGrRrrGGrRrr",
                                                        "yyrRrryyrRrr",
                                                        "GGrRrryyrRrr",
                                                        "GGrRrrRrrRrr",
                                                        "yyrRrrGGrRrr",
                                                        "RrrRrrGGrRrr",
                                                        "rrrrrrrrrrrr"}; //manually copied
from output of print_AllPhases function

//All phase status description
//1) if any light is yellow status = "Y"
//2) if only 0 or 1 phases are green, then status = "R"
//3) if 2 phases are green, then status = "G"
const char *ALLPHASESSTATUS[49] = {"G", "Y", "Y", "R", "Y", "R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "G", "Y", "Y", "R", "Y",
"R",
                                                                "R"};

//The currentPhaseIndex is the row index of NEXTPHASES
int NEXTPHASES[49][8] = {{0, 1, 2, 4},
                                                {48},
                                                {3},
```

```
                                                    {6},
                                                    {5},
                                                    {12},
                                                    {6, 7, 8, 10},
                                                    {48},
                                                    {9},
                                                    {0},
                                                    {11},
                                                    {18},
                                                    {12, 13, 14, 16},
                                                    {48},
                                                    {15},
                                                    {18},
                                                    {17},
                                                    {0},
                                                    {18, 19, 20, 22},
                                                    {48},
                                                    {21},
                                                    {12},
                                                    {23},
                                                    {6},
                                                    {24, 25, 26, 28},
                                                    {48},
                                                    {27},
                                                    {30},
                                                    {29},
                                                    {36},
                                                    {30, 31, 32, 34},
                                                    {48},
                                                    {33},
                                                    {24},
                                                    {35},
                                                    {42},
                                                    {36, 37, 38, 40},
                                                    {48},
                                                    {39},
                                                    {42},
                                                    {41},
                                                    {24},
                                                    {42, 43, 44, 46},
                                                    {48},
                                                    {45},
                                                    {36},
                                                    {47},
                                                    {30},
                                                    {0, 6, 12, 18, 24, 30, 36, 42}};
//maybe include 48 here as well (ALLRED can stay ALLRED for more than one second)

//Adaptive/Optimal Signal Control Parameters
int minimumGreenDuration = 8;
int maximumGreenDuration = 65; //set to be slightly larger than the time used for
a queue to clear a link adjacent to the intersection (assumming that the link is
fully loaded when the light turns green, and ignoring overflow into subsequent
links)
```

```
typedef struct intersection_profile INTERSECTION_DATA;
struct intersection_profile
{
	float phase_start; //in seconds, wrt to paramics time elapsed
	int currentPhaseIndex; //default starting phase is ONEFIVEGREEN
	int currentPhaseDuration; //= minimumGreenDuration; //default starting
duration is equal to min green (the idea behind a larger min green is to avoid the
partial queue discharge effect)

	VEHICLE* queueTailIDs[16]; //16 lanes for this particular network, change
to accomodate the total number of incoming lanes to the intersection, NOTE: can
change this later to be a property of the intersection, to accomodate multiple
intersections
	Bool queuesDischarged; //initialized to true for the sake of the first
phase of the simulation (in which there are no queues),  remember to set and reset
appropriately (set to false at the start of the second phase of the simulation)

	//variables for calculating and storing aging factor
	int lastPhaseIndex; //phase index of the time step immediately prior to the
current time step, NOTE: can update this as lastPhaseIndex = currentPhaseIndex at
the beginning of each time step in (qpx_CLK_startOfSimLoop())
	float timeLastServed[8]; //absolute time when the individual movement was
last served, NOTE: timeLastServed is absolute time when the movement was last
served (with a green light) and covers movements: 1, 2, 3, ...8
	float agingTime[8]; //size is 8 due to number of individual phases, units
is seconds, NOTE: aging time is time elapsed since the movement was last served
and covers movements: 1, 2, 3, ...8
	float agingFactor[8]; //aging factor muliplies queue length to modulate
fairness, based on elapsed time, NOTE: aging factors refers to dual phases: 1/5,
1/6, 2/5, 2/6, 3/7, 3/8, 4/7, 4/8

	//declare pointers for all links associated with the intersection
	LINK* WEST_in; //check if this works
	LINK* NORTH_in;
	LINK* EAST_in;
	LINK* SOUTH_in;

	LINK* WEST_out;
	LINK* NORTH_out;
	LINK* EAST_out;
	LINK* SOUTH_out;

	//drawing/statistics variables
	float PhaseCurrentDelay[8]; //used to hold current delay values of all 8
dual phases (sum of current delays of vehicles on all the links associated with a
particular phase)
	float circleCenterX; //X-coordinate for phase indicator
	float circleCenterY; //Y-coordinate for phase indicator
	float rectangleCenterX; //X-coordinate for max phase-length string
	float rectangleCenterY; //Y-coordinate for max phase-length string

	char s0[100], s1[100], s2[100], s3[100], s4[100], s5[100], s6[100],
s7[100], s8[100];
```

```
	char s9[100];

	int maxGreenReached; // test variable to determine what the longest green
duration during the simulation is
};

INTERSECTION_DATA intersection[3]; //NOTE: change length of array to accomodate
number of intersections along the corridor in the network
//int numberOfIntersections = ARRAYLENGTH(intersection); //3; //currently intended
for use on corridor

//NOTE: use node names of signals in corridor in the following array:
const char *INTERSECTIONIDS[3] = {"5", "m5", "n5"}; //for 3-intersection corridor
int numberOfIntersections = ARRAYLENGTH(INTERSECTIONIDS); //3; //currently
intended for use on corridor

//Drawing variables for AdSC
//char s0[100], s1[100], s2[100], s3[100], s4[100], s5[100], s6[100], s7[100],
s8[100];
//char s9[100];
Bool stringsInitialized = PFALSE; //false if the above strings are not initialized
//float PhaseCurrentDelay[8] = {0}; //used to hold current delay values of all 8
dual phases, (make this a field of the struct for each intersection in the
network)

void print_AllPhases()
{
	//NOTE: the ALLPHASES array contains 49 combinations of individual phases,
if additional phases are desired, the call to this function should be uncommented
	//	This function prints All Phases in the Paramics information
browser.  Copy-paste the text into the ALLPHASES array to complete the procedure.

	//Method 2: works
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN, FIVEGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", ONEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,FIVEGREEN));
	qps_GUI_printf("\"%s\", \n", FIVEGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,SIXGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEGREEN,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", ONEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(ONEYELLOW,SIXGREEN));
	qps_GUI_printf("\"%s\", \n", SIXGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,FIVEGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,FIVEYELLOW));
	qps_GUI_printf("\"%s\", ", TWOGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,FIVEGREEN));
	qps_GUI_printf("\"%s\", \n", FIVEGREEN);
```

```
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,SIXGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOGREEN,SIXYELLOW));
	qps_GUI_printf("\"%s\", ", TWOGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(TWOYELLOW,SIXGREEN));
	qps_GUI_printf("\"%s\", \n", SIXGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,SEVENGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,SEVENYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,SEVENYELLOW));
	qps_GUI_printf("\"%s\", ", THREEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,SEVENGREEN));
	qps_GUI_printf("\"%s\", \n", SEVENGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,EIGHTGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,EIGHTYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEGREEN,EIGHTYELLOW));
	qps_GUI_printf("\"%s\", ", THREEGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(THREEYELLOW,EIGHTGREEN));
	qps_GUI_printf("\"%s\", \n", EIGHTGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,SEVENGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,SEVENYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,SEVENYELLOW));
	qps_GUI_printf("\"%s\", ", FOURGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,SEVENGREEN));
	qps_GUI_printf("\"%s\", \n", SEVENGREEN);

	qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,EIGHTGREEN));
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,EIGHTYELLOW));
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURGREEN,EIGHTYELLOW));
	qps_GUI_printf("\"%s\", ", FOURGREEN);
	qps_GUI_printf("\"%s\", ", CombinePhases(FOURYELLOW,EIGHTGREEN));
	qps_GUI_printf("\"%s\", \n", EIGHTGREEN);

	qps_GUI_printf("\"%s\" \n", ALLRED);
}

int index_from_ID(char *intersectionID)
{
	int i;

	for(i = 0; i < numberOfIntersections; i++)
	{
		if (intersectionID == INTERSECTIONIDS[i])
		{
			return i;
		}
	}
	return -1; //error occurs
}

void InitializeIntersections()
{
```

```
	int i, j;

	for (i = 0; i < numberOfIntersections; i++)
	{
		intersection[i].currentPhaseDuration = 8; //= minimumGreenDuration;
//default starting duration is equal to min green (the idea behind a larger min
green is to avoid the partial queue discharge effect)
		intersection[i].currentPhaseIndex = 0; //default starting phase is
ONEFIVEGREEN
		intersection[i].phase_start = 0.0; //in seconds, wrt to paramics
time elapsed
		intersection[i].queuesDischarged = PTRUE; //initialized to true for
the sake of the first phase of the simulation (in which there are no queues),
remember to set and reset appropriately (set to false at the start of the second
phase of the simulation)
		//intersection[i].queueTailIDs[16]; //no initialization needed here
		intersection[i].lastPhaseIndex = 0;
		intersection[i].maxGreenReached = 8; // test variable to determine
what the longest green duration during the simulation is
		for (j = 0; j < 8; j++)
		{
			intersection[i].timeLastServed[j] = 0;
			intersection[i].agingTime[j] = 0;
			intersection[i].agingFactor[j] = 1;

			intersection[i].PhaseCurrentDelay[j] = 0;
		}

		//initialize intersection links
		if (i == 0)
		{
			intersection[i].WEST_in = qpg_NET_link("11:5"); //this set of
link pointers will only work for the following networks (and copies based on those
networks): "Isolated_CustomAdaptive_Four_way_Intersection_test1",
"3_IntersectionCorridor_CustomAdaptive_12matrices"
			intersection[i].NORTH_in = qpg_NET_link("13:5");
			intersection[i].EAST_in = qpg_NET_link("10:5");
			intersection[i].SOUTH_in = qpg_NET_link("12:5");

			intersection[i].WEST_out = qpg_NET_link("5:11");
			intersection[i].NORTH_out = qpg_NET_link("5:13");
			intersection[i].EAST_out = qpg_NET_link("5:10");
			intersection[i].SOUTH_out = qpg_NET_link("5:12");
		}
		else if (i == 1)
		{
			intersection[i].WEST_in = qpg_NET_link("m11:m5"); //this set
of link pointers will only work for the 3-intersection corridor network:
"3_IntersectionCorridor_CustomAdaptive_12matrices"
			intersection[i].NORTH_in = qpg_NET_link("m13:m5");
			intersection[i].EAST_in = qpg_NET_link("m10:m5");
			intersection[i].SOUTH_in = qpg_NET_link("m12:m5");

			intersection[i].WEST_out = qpg_NET_link("m5:m11");
```

```
                        intersection[i].NORTH_out = qpg_NET_link("m5:m13");
                        intersection[i].EAST_out = qpg_NET_link("m5:m10");
                        intersection[i].SOUTH_out = qpg_NET_link("m5:m12");
                }
                else if (i == 2)
                {
                        intersection[i].WEST_in = qpg_NET_link("n11:n5"); //this set
of link pointers will only work for the 3-intersection corridor network:
"3_IntersectionCorridor_CustomAdaptive_12matrices"
                        intersection[i].NORTH_in = qpg_NET_link("n13:n5");
                        intersection[i].EAST_in = qpg_NET_link("n10:n5");
                        intersection[i].SOUTH_in = qpg_NET_link("n12:n5");

                        intersection[i].WEST_out = qpg_NET_link("n5:n11");
                        intersection[i].NORTH_out = qpg_NET_link("n5:n13");
                        intersection[i].EAST_out = qpg_NET_link("n5:n10");
                        intersection[i].SOUTH_out = qpg_NET_link("n5:n12");
                }
        }

        //drawing code initializations
        for (i = 0; i < numberOfIntersections; i++)
        {
                //initialize drawing code variables
                if (i == 0)
                {
                        intersection[0].circleCenterX = -113.36f; //units are meters
                        intersection[0].circleCenterY = 631.52f;
                        intersection[0].rectangleCenterX = 173.36f;
                        intersection[0].rectangleCenterY = 548.64f;
                }
                else if (i == 1)
                {
                        //2414 ft between intersections = 735.7872 meters
                        intersection[1].circleCenterX = -113.36f - 735.79; //units are
meters, y-offset for network "3_IntersectionCorridor_CustomAdaptive_12matrices"
appears to be +186.82 m, or invert x and subtract 39.9 m
                        intersection[1].circleCenterY = 631.52f;
                        intersection[1].rectangleCenterX = 173.36f - 735.79;
                        intersection[1].rectangleCenterY = 548.64f;
                }
                else if (i == 2)
                {
                        intersection[2].circleCenterX = -113.36 - 2*735.79f; //units
are meters
                        intersection[2].circleCenterY = 631.52f;
                        intersection[2].rectangleCenterX = 173.36 - 2*735.79f;
                        intersection[2].rectangleCenterY = 548.64f;
                }

                //acccount for network offset
                translatePoint(&intersection[i].circleCenterX,
&intersection[i].circleCenterY);
```

```
		translatePoint(&intersection[i].rectangleCenterX,
&intersection[i].rectangleCenterY);
	}
}

void qpx_NET_postOpen()
{
	int i, j, k, md, q;
	int reg[13] = {11,21,31,32,41,42,43,51,52,53,54,61,62}; // source types
	char *path, *outPath;
	char inPath[150] = "";
	FILE *fin;

	//Drawing initializations
	/** These are set here so as to optimise the drawing code. */

	/* Obtain the bounding box of the network. If the point (0,0) is passed to
	* qps_DRW_filledCircle, then the circle will be drawn at (llx, lly)....
	*/
	qpg_POS_network(&llx, &lly, &urx, &ury);

	/**
	* We need to correct x coordinates if the network is right hand drive.
	*/
	rhd = qpg_CFG_driveOnRight(); //this should return true; //NOTE this
function must be called before initializing the intersections
(InitializeIntersections())
	InitializeIntersections();

	n[0] = qpg_NET_nodeByIndex(5); //check if this index works

	//attempt to set dummy phase (only works when links next to zones have
shortened signposts, no idea why, bizarre,...just use the shortened signposts (820
down to 520 ft)
	//apply this to corridor, is this really necessary?
	for (q = 0; q < numberOfIntersections; q++)
	{
		qps_LNK_priority(intersection[q].NORTH_in, intersection[q].WEST_out,
APIPRI_MAJOR);  //4
		qps_LNK_priority(intersection[q].NORTH_in,
intersection[q].SOUTH_out, APIPRI_MAJOR); //4
		qps_LNK_priority(intersection[q].NORTH_in, intersection[q].EAST_out,
APIPRI_MAJOR);  //7

		qps_LNK_priority(intersection[q].EAST_in, intersection[q].NORTH_out,
APIPRI_MAJOR);  //6
		qps_LNK_priority(intersection[q].EAST_in, intersection[q].WEST_out,
APIPRI_MAJOR);   //6
		qps_LNK_priority(intersection[q].EAST_in, intersection[q].SOUTH_out,
APIPRI_MAJOR);  //1

		qps_LNK_priority(intersection[q].SOUTH_in, intersection[q].EAST_out,
APIPRI_MAJOR);  //8
```

```
            qps_LNK_priority(intersection[q].SOUTH_in,
intersection[q].NORTH_out, APIPRI_MAJOR); //8
            qps_LNK_priority(intersection[q].SOUTH_in, intersection[q].WEST_out,
APIPRI_MAJOR);  //3

            qps_LNK_priority(intersection[q].WEST_in, intersection[q].SOUTH_out,
APIPRI_MAJOR);  //2
            qps_LNK_priority(intersection[q].WEST_in, intersection[q].EAST_out,
APIPRI_MAJOR);  //2
            qps_LNK_priority(intersection[q].WEST_in, intersection[q].NORTH_out,
APIPRI_MAJOR); //5
      }

      DELTA = 0.5*qpg_CFG_timeStep();
      path = qpg_NET_dataPath();
      outPath = qpg_NET_statsPath();
      strcpy(inPath, path);
      strcpy(out1Path, outPath);
      strcpy(out2Path, outPath);
      strcpy(out3Path, outPath);
      strcpy(out4Path, outPath);
      strcpy(out5Path, outPath);
      strcpy(out11Path, outPath);
      strcpy(out21Path, outPath);
      strcpy(out31Path, outPath);
      strcpy(out41Path, outPath);
      strcpy(out51Path, outPath);
      strcpy(out12Path, outPath);
      strcpy(out22Path, outPath);
      strcpy(out32Path, outPath);
      strcpy(out42Path, outPath);
      strcpy(out52Path, outPath);

      strcat(inPath, "/sourceTypes_2005.txt");
      strcat(out1Path, "/data_sbs.dat");
      strcat(out2Path, "/MOVES_sum.dat");
      strcat(out3Path, "/opMode.dat");
      strcat(out4Path, "/VSP.dat");
      strcat(out5Path, "/TT.dat");
      strcat(out11Path, "/data_sbs_e.dat");
      strcat(out21Path, "/MOVES_sum_e.dat");
      strcat(out31Path, "/opMode_e.dat");
      strcat(out41Path, "/VSP_e.dat");
      strcat(out51Path, "/TT_e.dat");
      strcat(out12Path, "/data_sbs_n.dat");
      strcat(out22Path, "/MOVES_sum_n.dat");
      strcat(out32Path, "/opMode_n.dat");
      strcat(out42Path, "/VSP_n.dat");
      strcat(out52Path, "/TT_n.dat");

      fin = fopen(inPath, "r");
      if (fin == NULL)
            qps_GUI_printf("Couldn't open file. (simulation started)");
      fscanf(fin, "%d,", &md); // md: opMode
```

```
	for (i = 0; i < 13; i++) // i: regClass
		for (j = 0; j < 41; j++) // j: opMode
			if (md == j)
			{
				fscanf(fin, "%f,%f,%f,%f,%f,%f", &moves[reg[i]][j][0],
&moves[reg[i]][j][1], &moves[reg[i]][j][2], &moves[reg[i]][j][3],
&moves[reg[i]][j][4], &moves[reg[i]][j][5]);
				fscanf(fin, "%d,", &md);
			}
	for (j = 0; j < 41; j++) {
		for (k = 0; k <= 5; k++) {
			moves[20][j][k] = moves[21][j][k];
		}
	}
	fclose(fin);

	qps_GUI_printf("NET_postOpen entered\n");

	qps_GUI_printf("Connected Vehicle Adaptive Signal Control Application for
corridor of %d intersections\n", numberOfIntersections);
}

void qpx_DRW_modelView()
{
	//Adaptive/Optimal Signal Control Test Drawing Code
	int i = 0;
	float currentTime = qpg_CFG_simulationTime(); //elapsed simulation time in
seconds
	float yOffset = 0;
	float testSpeed = 60;
	float tempFloat = 0.0f;
	VEHICLE* temp_queueTailIDs[16];
	VEHICLE* temp_queueTailIDs2[16];

	//update data once a second (to prevent large overhead)
	if (fabs(currentTime - (int)currentTime) < DELTA)
	{
		//set strings
		/*sprintf(s0, "Combined Queue Lengths by NEMA Phase: ");
		sprintf(s1, "1/5: %d ", Find_Phase_Queue_Length(5, ALLPHASES[0],
&temp_queueTailIDs));
		sprintf(s2, "1/6: %d ", Find_Phase_Queue_Length(5, ALLPHASES[6],
&temp_queueTailIDs));
		sprintf(s3, "2/5: %d ", Find_Phase_Queue_Length(5, ALLPHASES[12],
&temp_queueTailIDs));
		sprintf(s4, "2/6: %d ", Find_Phase_Queue_Length(5, ALLPHASES[18],
&temp_queueTailIDs));
		sprintf(s5, "3/7: %d ", Find_Phase_Queue_Length(5, ALLPHASES[24],
&temp_queueTailIDs));
		sprintf(s6, "3/8: %d ", Find_Phase_Queue_Length(5, ALLPHASES[30],
&temp_queueTailIDs));
		sprintf(s7, "4/7: %d ", Find_Phase_Queue_Length(5, ALLPHASES[36],
&temp_queueTailIDs));
```

```
		sprintf(s8, "4/8: %d ", Find_Phase_Queue_Length(5, ALLPHASES[42],
&temp_queueTailIDs));*/

		/*sprintf(s0, "Combined Queue Lengths by NEMA Phase, QL x
AgingFactor");
		sprintf(s1, "1/5: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[0], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[0],
&temp_queueTailIDs)*agingFactor[0]);
		sprintf(s2, "1/6: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[6], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[6],
&temp_queueTailIDs)*agingFactor[1]);
		sprintf(s3, "2/5: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[12], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[12],
&temp_queueTailIDs)*agingFactor[2]);
		sprintf(s4, "2/6: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[18], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[18],
&temp_queueTailIDs)*agingFactor[3]);
		sprintf(s5, "3/7: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[24], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[24],
&temp_queueTailIDs)*agingFactor[4]);
		sprintf(s6, "3/8: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[30], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[30],
&temp_queueTailIDs)*agingFactor[5]);
		sprintf(s7, "4/7: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[36], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[36],
&temp_queueTailIDs)*agingFactor[6]);
		sprintf(s8, "4/8: %3d,  %5.2f ", Find_Phase_Queue_Length(5,
ALLPHASES[42], &temp_queueTailIDs), Find_Phase_Queue_Length(5, ALLPHASES[42],
&temp_queueTailIDs)*agingFactor[7]);*/

		for (i = 0; i < numberOfIntersections; i++)
		{
			//this section is primarily intended for use with OPTIMIZE7()
			sprintf(intersection[i].s0, "Combined Queue Lengths by NEMA
Phase, Current Delay");
			sprintf(intersection[i].s1, "1/5: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[0], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[0]);
			sprintf(intersection[i].s2, "1/6: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[6], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[1]);
			sprintf(intersection[i].s3, "2/5: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[12], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[2]);
			sprintf(intersection[i].s4, "2/6: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[18], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[3]);
			sprintf(intersection[i].s5, "3/7: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[24], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[4]);
			sprintf(intersection[i].s6, "3/8: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[30], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[5]);
```

```
                    sprintf(intersection[i].s7, "4/7: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[36], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[6]);
                    sprintf(intersection[i].s8, "4/8: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[42], &temp_queueTailIDs),
intersection[i].PhaseCurrentDelay[7]);

                    ////this section is primarily intended for use with
OPTIMIZE6()
                    //sprintf(intersection[i].s0, "Combined Queue Lengths by NEMA
Phase, QL x AgingFactor");
                    //sprintf(intersection[i].s1, "1/5: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[0], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[0],
&temp_queueTailIDs)*intersection[i].agingFactor[0]);
                    //sprintf(intersection[i].s2, "1/6: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[6], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[6],
&temp_queueTailIDs)*intersection[i].agingFactor[1]);
                    //sprintf(intersection[i].s3, "2/5: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[12], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[12],
&temp_queueTailIDs)*intersection[i].agingFactor[2]);
                    //sprintf(intersection[i].s4, "2/6: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[18], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[18],
&temp_queueTailIDs)*intersection[i].agingFactor[3]);
                    //sprintf(intersection[i].s5, "3/7: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[24], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[24],
&temp_queueTailIDs)*intersection[i].agingFactor[4]);
                    //sprintf(intersection[i].s6, "3/8: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[30], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[30],
&temp_queueTailIDs)*intersection[i].agingFactor[5]);
                    //sprintf(intersection[i].s7, "4/7: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[36], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[36],
&temp_queueTailIDs)*intersection[i].agingFactor[6]);
                    //sprintf(intersection[i].s8, "4/8: %3d,  %5.2f ",
Find_Phase_Queue_Length(i, ALLPHASES[42], &temp_queueTailIDs),
Find_Phase_Queue_Length(i, ALLPHASES[42],
&temp_queueTailIDs)*intersection[i].agingFactor[7]);
             }

             ////blinking problem solved by moving Find_Phase_Current_Delay
calculation to qpx_CLK_endOfSimLoop() function
             ////tempFloat = Find_Phase_Current_Delay0(5, ALLPHASES[12]); //test
this today
             ////qps_GUI_printf("total current delay on 2/5 = %f \n", tempFloat);
             ////qps_GUI_printf("total current delay on 2/5 = %f \n",
PhaseCurrentDelay[2]);
             //sprintf(s0, "Current Delay by NEMA Phase");
```

```
		//sprintf(s1, "1/5: %5.2f ", PhaseCurrentDelay[0]); //NOTE:
PhaseCurrentDelay is updated in the qpx_CLK_endOfSimLoop() function once a second
		//sprintf(s2, "1/6: %5.2f ", PhaseCurrentDelay[1]);
		//sprintf(s3, "2/5: %5.2f ", PhaseCurrentDelay[2]);
		//sprintf(s4, "2/6: %5.2f ", PhaseCurrentDelay[3]);
		//sprintf(s5, "3/7: %5.2f ", PhaseCurrentDelay[4]);
		//sprintf(s6, "3/8: %5.2f ", PhaseCurrentDelay[5]);
		//sprintf(s7, "4/7: %5.2f ", PhaseCurrentDelay[6]);
		//sprintf(s8, "4/8: %5.2f ", PhaseCurrentDelay[7]);

		for (i = 0; i < numberOfIntersections; i++)
		{
			sprintf(intersection[i].s9, "Maximum Green Reached: %d ",
intersection[i].maxGreenReached); //fix maxGreenReached later to be intersection
specific
		}

		stringsInitialized = PTRUE;
	}
	else if (stringsInitialized == PFALSE) //then initialized strings with 0
values for queue lengths
	{
		//set strings
		/*sprintf(s0, "Combined Queue Lengths by NEMA Phase: ");
		sprintf(s1, "1/5: %d ", 0);
		sprintf(s2, "1/6: %d ", 0);
		sprintf(s3, "2/5: %d ", 0);
		sprintf(s4, "2/6: %d ", 0);
		sprintf(s5, "3/7: %d ", 0);
		sprintf(s6, "3/8: %d ", 0);
		sprintf(s7, "4/7: %d ", 0);
		sprintf(s8, "4/8: %d ", 0);*/

		for (i = 0; i < numberOfIntersections; i++)
		{
			sprintf(intersection[i].s0, "Combined Queue Lengths by NEMA
Phase, QL x AgingFactor");
			sprintf(intersection[i].s1, "1/5: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s2, "1/6: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s3, "2/5: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s4, "2/6: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s5, "3/7: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s6, "3/8: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s7, "4/7: %3d,  %5.2f ", 0, 0);
			sprintf(intersection[i].s8, "4/8: %3d,  %5.2f ", 0, 0);

			sprintf(intersection[i].s9, "Maximum Green Reached: %d ", 0);
		}

		stringsInitialized = PTRUE;
	}
	//translatePoint(&circleCentreX, &circleCentreY);
	if (currentTime > 1.5)
	{
```

```
		//NOTE: the following drawing code is called once every time step
(otherwise the code will not be displayed
		/**
		* Translate the coordinates of the circle centre, and correct for
right hand drive (if needed).
		*/
		for (i = 0; i < numberOfIntersections; i++)
		{
			//translatePoint(&(intersection[i].circleCenterX),
&(intersection[i].circleCenterY));
			//translatePoint(&(intersection[i].rectangleCenterX),
&(intersection[i].rectangleCenterY));

			qps_DRW_colourRGB(1, 1, 1);
			//NOTE: x is flipped
			qps_DRW_stringXY(intersection[i].s0,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY + 5.0, 10.0);
//first draw string, NOTE: text height is in meters
			qps_DRW_stringXY(intersection[i].s1,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 5.0, 10.0);
			qps_DRW_stringXY(intersection[i].s2,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 15.0, 10.0);
			qps_DRW_stringXY(intersection[i].s3,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 25.0, 10.0);
			qps_DRW_stringXY(intersection[i].s4,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 35.0, 10.0);
			qps_DRW_stringXY(intersection[i].s5,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 45.0, 10.0);
			qps_DRW_stringXY(intersection[i].s6,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 55.0, 10.0);
			qps_DRW_stringXY(intersection[i].s7,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 65.0, 10.0);
			qps_DRW_stringXY(intersection[i].s8,
intersection[i].circleCenterX + 30.0, intersection[i].circleCenterY - 75.0, 10.0);

			qps_DRW_stringXY(intersection[i].s9,
intersection[i].rectangleCenterX, intersection[i].rectangleCenterY, 10.0);

			//Draw the circle
			qps_DRW_colourRGB(0, 1, 0);
			//set position of circle depending on active phase
			if (intersection[i].currentPhaseIndex == 0)
			{
				yOffset = -5.0;
			}
			else if (intersection[i].currentPhaseIndex == 6)
			{
				yOffset = -15.0;
			}
			else if (intersection[i].currentPhaseIndex == 12)
			{
				yOffset = -25.0;
			}
			else if (intersection[i].currentPhaseIndex == 18)
```

```
                        {
                                yOffset = -35.0;
                        }
                        else if (intersection[i].currentPhaseIndex == 24)
                        {
                                yOffset = -45.0;
                        }
                        else if (intersection[i].currentPhaseIndex == 30)
                        {
                                yOffset = -55.0;
                        }
                        else if (intersection[i].currentPhaseIndex == 36)
                        {
                                yOffset = -65.0;
                        }
                        else if (intersection[i].currentPhaseIndex == 42)
                        {
                                yOffset = -75.0;
                        }

                        if (ALLPHASESSTATUS[intersection[i].currentPhaseIndex] == "G")
                        {
                                qps_DRW_filledCircle(intersection[i].circleCenterX +
40.0, intersection[i].circleCenterY + yOffset + 5.0, 0, 5); //then draw circle,
NOTE: radius is in meters
                        }

                        //Draw the rectangle
                        qps_DRW_colourRGB(1, 0, 0);
                        qps_DRW_filledRectangleXY(intersection[i].rectangleCenterX +
2.5, intersection[i].rectangleCenterY - 2.5, intersection[i].rectangleCenterX -
140.0, intersection[i].rectangleCenterY + 12.5); //float bl_x, float bl_y, float
tr_x, float tr_y
                }

                //qps_GUI_printf("DRW_modelView entered\n");
        }
}

int opMode(float vsp, float speed, float acc)
{
        speed = speed*INDEX;
        acc = acc*INDEX;
        if (acc <= -2) return 0;

        if (speed < 1 && speed >= -1) return 1;
        if (speed < 25 && speed >= 0) //this second value should be 1, (error in
Haitao's code, never reached though, due to previous statement)
        {
                if (vsp < 0) return 11;
                if (vsp < 3) return 12;
                if (vsp < 6) return 13;
                if (vsp < 9) return 14;
                if (vsp < 12) return 15;
```

```
                return 16;}
        else if (speed < 50) {
                if (vsp < 0) return 21;
                if (vsp < 3) return 22;
                if (vsp < 6) return 23;
                if (vsp < 9) return 24;
                if (vsp < 12) return 25;
                if (vsp < 18) return 27;
                if (vsp < 24) return 28;
                if (vsp < 30) return 29;
                return 30;}
        else {
                if (vsp < 6) return 33;
                if (vsp < 12) return 35;
                if (vsp < 18) return 37;
                if (vsp < 24) return 38;
                if (vsp < 30) return 39;
                return 40;}
}

void updateVehAttributes(VEHICLE_DATA* myVeh)
{
        if(myVeh->sourceType == 11)
        {
                myVeh->A = 0.0251;
         myVeh->B = 0;
         myVeh->C = 0.000315;
         myVeh->M = 0.285;
                myVeh->f = 0.285;
                //myVeh->regClass = 10;
        }
        else if(myVeh->sourceType == 21 || myVeh->sourceType == 20)
        {
                myVeh->A = 0.156461;
         myVeh->B = 0.002002;
         myVeh->C = 0.000493;
         myVeh->M = 1.4788;
                myVeh->f = 1.4788;
                //myVeh->regClass = 20;
        }
        else if(myVeh->sourceType == 31)
        {
                myVeh->A = 0.22112;
         myVeh->B = 0.002838;
         myVeh->C = 0.000698;
         myVeh->M = 1.86686;
                myVeh->f = 1.86686;
                //myVeh->regClass = 30;
        }
        else if(myVeh->sourceType == 32)
        {
                myVeh->A = 0.235008;
         myVeh->B = 0.003039;
         myVeh->C = 0.000748;
```

```
 myVeh->M = 2.05979;
       myVeh->f = 2.05979;
       //myVeh->regClass = 30;
}
else if(myVeh->sourceType == 41)
{
       myVeh->A = 1.29515;
 myVeh->B = 0;
 myVeh->C = 0.003715;
 myVeh->M = 19.5937;
       myVeh->f = 17.1;
       //myVeh->regClass = 48;
}
else if(myVeh->sourceType == 42)
{
       myVeh->A = 1.0944;
 myVeh->B = 0;
 myVeh->C = 0.003587;
 myVeh->M = 16.556;
       myVeh->f = 17.1;
       //myVeh->regClass = 48;
}
else if(myVeh->sourceType == 43)
{
       myVeh->A = 0.746718;
 myVeh->B = 0;
 myVeh->C = 0.002176;
 myVeh->M = 9.06989;
       myVeh->f = 17.1;
       //myVeh->regClass = 46;
}
else if(myVeh->sourceType == 51)
{
       myVeh->A = 1.41705;
 myVeh->B = 0;
 myVeh->C = 0.003572;
 myVeh->M = 20.6845;
       myVeh->f = 17.1;
       //myVeh->regClass = 47;
}
else if(myVeh->sourceType == 52)
{
       myVeh->A = 0.561933;
 myVeh->B = 0;
 myVeh->C = 0.001603;
 myVeh->M = 7.64159;
       myVeh->f = 17.1;
       //myVeh->regClass = 42;
}
else if(myVeh->sourceType == 53)
{
       myVeh->A = 0.498699;
 myVeh->B = 0;
 myVeh->C = 0.001474;
```

```
        myVeh->M = 6.25047;
              myVeh->f = 17.1;
              //myVeh->regClass = 41;
       }
       else if(myVeh->sourceType == 54)
       {
              myVeh->A = 0.617371;
        myVeh->B = 0;
        myVeh->C = 0.002105;
        myVeh->M = 6.73483;
              myVeh->f = 17.1;
              //myVeh->regClass = 42;
       }
       else if(myVeh->sourceType == 61)
       {
              myVeh->A = 1.96354;
        myVeh->B = 0;
        myVeh->C = 0.004031;
        myVeh->M = 29.3275;
              myVeh->f = 17.1;
              //myVeh->regClass = 47;
       }
       else //if(myVeh->sourceType == 62)
       {
              myVeh->A = 2.08126;
        myVeh->B = 0;
        myVeh->C = 0.004188;
        myVeh->M = 31.4038;
              myVeh->f = 17.1;
              //myVeh->regClass = 47;
       }
}

void  qpx_VHC_release(VEHICLE* Vp)
{
       VEHICLE_DATA *myVeh = calloc(1, sizeof(VEHICLE_DATA));
       double rn;
       float x5 = 0; // used for destination node location
       float y5 = 0; // used for destination node location
       float z5 = 0; //unused
       LINK* link = qpg_VHC_link(Vp);           // pointer of the current link
(used for ESH)
       counter_released++;

       // initialize data for MOVES model
       myVeh->ID = qpg_VHC_uniqueID(Vp);
       myVeh->sourceType = qpg_VHC_type(Vp); // emission tables directly related
to source types
       myVeh->theta = 0;
       myVeh->first2sec = 1;
       myVeh->origin = qpg_ZNE_externalIndex(qpg_VHC_origin(Vp));
       myVeh->dest = qpg_ZNE_externalIndex(qpg_VHC_destination(Vp));
       myVeh->tripDist = 0;//qpg_RTR_distanceRemaining(qpg_VHC_link(Vp), Vp);
//this function is bugged, and is not working with the on/off ramp network
```

```
	//retrieve destination node position (for CACC)
	qpg_POS_node(qpg_LNK_nodeStart(qpg_ZNE_link(qpg_NET_zone(myVeh->dest), 1)),
&x5, &y5, &z5);
	myVeh->x_destNode = x5;
	myVeh->y_destNode = y5;

	myVeh->total_energy = 0; //default initialization, aggregated when vehicle
is in the network

	// initialize variables for Adaptive Signal Control
	myVeh->currentNetworkDelay = 0;

	updateVehAttributes(myVeh);
	qps_VHC_userdata(Vp, (VEHICLE_DATA *)myVeh);

	//qps_GUI_printf("Vehicle %d trip %d -> %d on link %s\n",
qpg_VHC_uniqueID(Vp), qpg_VHC_origin(Vp), qpg_VHC_destination(Vp),
qpg_LNK_name(qpg_VHC_link(Vp)));

	if ((myVeh->origin == 1 && myVeh->dest == 2) || (myVeh->origin == 2 &&
myVeh->dest == 1)) //if vehicle is on the coordinated phase
	{
		counter_released_c++;
	}
	else //if vehicle is not part of the coordinated phase
	{
		counter_released_nc++;
	}

	//zone test:
	if (myVeh->origin == 1 && myVeh->dest == 2)
	{
		qps_DRW_vehicleColour(Vp, 4294967040); //cyan 4294967040
(0xFFFFFF00)
		//qps_GUI_printf("Zone 5 internal index = %d, external index = %d
\n", myVeh->origin, qpg_ZNE_externalIndex(myVeh->origin));

	}
	else if (myVeh->origin == 2 && myVeh->dest == 1)
	{
		qps_DRW_vehicleColour(Vp, 4278255360); //light green
		//qps_GUI_printf("Zone 6 internal index = %d, external index = %d
\n", myVeh->origin, qpg_ZNE_externalIndex(myVeh->origin));
	}
}

void calEmissions(VEHICLE_DATA* myVeh, float currTime, float vel_c, int linkid)
{
	int i;
	// shift 4-second data
	for (i = 0; i < 3; i++)
	{
		myVeh->vel[i] = myVeh->vel[i+1];
```

```
		myVeh->acc[i] = myVeh->acc[i+1];
		myVeh->linkID[i] = myVeh->linkID[i+1];
		myVeh->VSP[i] = myVeh->VSP[i+1];
		myVeh->mode[i] = myVeh->mode[i+1];
		myVeh->tm[i] = myVeh->tm[i+1];
		myVeh->energy[i] = myVeh->energy[i+1];
		myVeh->CO2[i] = myVeh->CO2[i+1];
		myVeh->CO[i] = myVeh->CO[i+1];
		myVeh->HC[i] = myVeh->HC[i+1];
		myVeh->NOx[i] = myVeh->NOx[i+1];
		myVeh->PM[i] = myVeh->PM[i+1];
	}
	myVeh->tm[3] = currTime;
	myVeh->linkID[3] = linkid;
	myVeh->vel[3] = vel_c;
	myVeh->acc[2] = (myVeh->vel[3] - myVeh->vel[1])/2;
	myVeh->VSP[2] = (myVeh->A*myVeh->vel[2] + myVeh->B*pow(myVeh->vel[2],2) +
myVeh->C*pow(myVeh->vel[2],3) + myVeh->M*(myVeh->acc[2] + GRAV*sin(myVeh-
>theta))*myVeh->vel[2])/myVeh->f;
	myVeh->mode[2] = opMode(myVeh->VSP[2], myVeh->vel[2], myVeh->acc[2]);
	if (myVeh->acc[0] < -1 && myVeh->acc[1] < -1 && myVeh->acc[2] < -1) myVeh-
>mode[2] = 0;
	modeBins[myVeh->mode[2]] += 1;
	myVeh->HC[2] = moves[myVeh->sourceType][myVeh->mode[2]][0];
	myVeh->CO[2] = moves[myVeh->sourceType][myVeh->mode[2]][1];
	myVeh->NOx[2] = moves[myVeh->sourceType][myVeh->mode[2]][2];
	myVeh->CO2[2] = moves[myVeh->sourceType][myVeh->mode[2]][3];
	myVeh->energy[2] = moves[myVeh->sourceType][myVeh->mode[2]][4];
	myVeh->PM[2] = moves[myVeh->sourceType][myVeh->mode[2]][5];

	//update overall statistics
	dist_sum += myVeh->vel[2];
	energy_sum += myVeh->energy[2];
	CO2_sum += myVeh->CO2[2];
	CO_sum += myVeh->CO[2];
	HC_sum += myVeh->HC[2];
	NOx_sum += myVeh->NOx[2];
	PM_sum += myVeh->PM[2];

	if ((myVeh->origin == 1 && myVeh->dest == 2) || (myVeh->origin == 2 &&
myVeh->dest == 1)) //update statistics for coordinated phase vehicles
	{
		dist_sum_c += myVeh->vel[2];
		energy_sum_c += myVeh->energy[2];
		CO2_sum_c += myVeh->CO2[2];
		CO_sum_c += myVeh->CO[2];
		HC_sum_c += myVeh->HC[2];
		NOx_sum_c += myVeh->NOx[2];
		PM_sum_c += myVeh->PM[2];
	}
	else //vehicle is not on the coordinated pahse  //update statistics for
non-coordinated phase vehicles
	{
		dist_sum_nc += myVeh->vel[2];
```

```
		energy_sum_nc += myVeh->energy[2];
		CO2_sum_nc += myVeh->CO2[2];
		CO_sum_nc += myVeh->CO[2];
		HC_sum_nc += myVeh->HC[2];
		NOx_sum_nc += myVeh->NOx[2];
		PM_sum_nc += myVeh->PM[2];
	}

	//aggregate individual vehicle energy
	myVeh->total_energy += myVeh->energy[2];
	myVeh->total_dist += myVeh->vel[2];
}

void SetRedYellowGreenState(int intersectionIndex, const char *phaseString)
{
	//This function sets the priorities of a 4-way intersection to mimic
adaptive signal control
	//Adaptive Signal Control Work-around:
	// MAJOR = Green    (G)
	// MEDIUM = Yellow  (y)
	// MINOR = RTOR     (R)
	// BARRED = Red     (r)

	int intIndex = intersectionIndex;
	int i;
	int priorityArray[12]; //holds priority integers (based on paramics API: 0
= Major, 1 = Medium, 2 = Minor, 3 = Barred) for all 12 movements of a 4-way
intersection
	//declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
	LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
	LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
	LINK* EAST_in_local = intersection[intIndex].EAST_in;
	LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

	LINK* WEST_out_local = intersection[intIndex].WEST_out;
	LINK* NORTH_out_local = intersection[intIndex].NORTH_out;
	LINK* EAST_out_local = intersection[intIndex].EAST_out;
	LINK* SOUTH_out_local = intersection[intIndex].SOUTH_out;

	//translate phaseString into priority array
	for (i = 0; i < strlen(phaseString); i++) //length of phaseString should be
12
	{
		if (phaseString[i] == 'G') //double-check the quotes
		{
			priorityArray[i] = APIPRI_MAJOR; //mimics green light
		}
		else if (phaseString[i] == 'y')
		{
			priorityArray[i] = APIPRI_MEDIUM; // mimics yellow light
		}
		else if (phaseString[i] == 'R')
		{
```

```
                        priorityArray[i] = APIPRI_MINOR; // mimics RTOR (right turn on
red)
                }
                else if (phaseString[i] == 'r')
                {
                        priorityArray[i] = APIPRI_BARRED; // mimics red light
                }
        }

        //set priorities                                          //NEMA phase
equivalent
        qps_LNK_priority(NORTH_in_local, WEST_out_local, priorityArray[0]);  //4
        qps_LNK_priority(NORTH_in_local, SOUTH_out_local, priorityArray[1]); //4
        qps_LNK_priority(NORTH_in_local, EAST_out_local, priorityArray[2]);  //7

        qps_LNK_priority(EAST_in_local, NORTH_out_local, priorityArray[3]);  //6
        qps_LNK_priority(EAST_in_local, WEST_out_local, priorityArray[4]);   //6
        qps_LNK_priority(EAST_in_local, SOUTH_out_local, priorityArray[5]);  //1

        qps_LNK_priority(SOUTH_in_local, EAST_out_local, priorityArray[6]);  //8
        qps_LNK_priority(SOUTH_in_local, NORTH_out_local, priorityArray[7]); //8
        qps_LNK_priority(SOUTH_in_local, WEST_out_local, priorityArray[8]);  //3

        qps_LNK_priority(WEST_in_local, SOUTH_out_local, priorityArray[9]);  //2
        qps_LNK_priority(WEST_in_local, EAST_out_local, priorityArray[10]);  //2
        qps_LNK_priority(WEST_in_local, NORTH_out_local, priorityArray[11]); //5

}

VEHICLE *GetQueueTailVp(LINK *incomingLink, int incomingLane, int
lane_queue_length1, LINK *permittedNextLink1, LINK *permittedNextLink2)
{
        //Finds the pointer of the first vehicle (starting from the rear of the
queue which has a correct next movement for the given lane
        //If no such vehicle is found, return value is NULL
        //parameter description:
        //     incomingLink: is the pointer to one of the links that permits
vehicles to enter the intersection
        //  incomingLane: is the lane index of the lane currently being examined
        //  lane_queue_length: is the number of vehicles currently on the lane-link
combination
        //  permittedNextLink#: refers to permitted next links based on the current
lane-link
        //            NOTE: currently this function supports up to two next links,
(2 next links indicates a shared movement)
        //                     If there is only one next link (no shared movement),
then the remaining next link pointers should be set to NULL
        VEHICLE *tempVp = qpg_LNK_vehicleTail(incomingLink, incomingLane); //used
in the process of finding the vehicle pointer of the correct vehicle tail
        int temp_lane_queue_length = lane_queue_length1; //used in the case where
the actual tail vehicle of the platoon is not in the correct lane for its intended
movement through the intersection
        Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found
```

```
	if (lane_queue_length1 > 0)
	{
		if (permittedNextLink2 != NULL)
		{
			if ((qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink1) || (qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink2)) //shared movement
			{
				return tempVp;
			}
			else if (temp_lane_queue_length == 1) //only one vehicle was
in the lane, use dummy queueTailID bc vehicle is in wrong lane
			{
				return NULL;
			}
			else //temp_lane_queue_length is >= 2 and actual queue tail
has wrong lane
			{
				while ((temp_lane_queue_length > 1) && (FOUND ==
PFALSE))
				{
					temp_lane_queue_length -= 1;
					tempVp = qpg_VHC_ahead(tempVp); //get the next
vehicle ahead in the queue
					if ((qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink1) || (qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink2)) //shared movement
					{
						return tempVp;
						FOUND = PTRUE; //unnecessary
					}
				}

				if (FOUND == PFALSE) //temp_lane_queue_length should =
1 at this point, if FOUND == PFALSE
				{
					//out_temp_queueTailIDs[0] = NULL; //check if
this works
					return NULL;
				}

			}
		}
		else //no shared movement, (only one next link is permitted)
		{
			if (qpg_LNK_exit(incomingLink, qpg_VHC_nextExit(tempVp)) ==
permittedNextLink1)
			{
				return tempVp;
			}
			else if (temp_lane_queue_length == 1) //only one vehicle was
in the lane, use dummy queueTailID bc vehicle is in wrong lane
			{
```

```
                                return NULL;
                        }
                        else //temp_lane_queue_length is >= 2 and actual queue tail
has wrong lane
                        {
                                while ((temp_lane_queue_length > 1) && (FOUND ==
PFALSE))
                                {
                                        temp_lane_queue_length -= 1;
                                        tempVp = qpg_VHC_ahead(tempVp); //get the next
vehicle ahead in the queue
                                        if (qpg_LNK_exit(incomingLink,
qpg_VHC_nextExit(tempVp)) == permittedNextLink1)
                                        {;
                                                return tempVp;
                                                FOUND = PTRUE; //unnecessary
                                        }
                                }

                                if (FOUND == PFALSE) //temp_lane_queue_length should =
1 at this point, if FOUND == PFALSE
                                {
                                        return NULL;
                                }

                        }
                }
        }
        else
        {
                return NULL;
        }
}

int *Find_All_Individual_Phase_Queue_Lengths(int intersectionIndex)
{
        //This function finds the total queue length for a given individual phase
(i.e. half of the NEMA dual phase), summing up over all lanes belonging to the
movement
        //NOTE: This function simply counts all vehicles on the lane-link closest
to intersection regardless of range or speed
        int intIndex = intersectionIndex;
        int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve
        static int individual_phase_queue_length[8] = {0};
        int *output;

        //declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
        LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
        LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
        LINK* EAST_in_local = intersection[intIndex].EAST_in;
        LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;
```

```
	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate combined queue length: sum over all lanes of the
prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)

	//1)Get individual lane queue lengths
	// NEMA Phase 4
	lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
	lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
	lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);

	// NEMA Phase 7
	lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);

	// NEMA Phase 6
	lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
	lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
	lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);

	// NEMA Phase 1
	lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);

	// NEMA Phase 8
	lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
	lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
	lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);

	// NEMA Phase 3
	lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);

	// NEMA Phase 2
	lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
	lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
	lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);

	// NEMA Phase 5
	lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);

	//2) aggregate lanes for individual movements
	individual_phase_queue_length[0] = lane_queue_length[7]; // NEMA Phase 1
	individual_phase_queue_length[1] = lane_queue_length[12] +
lane_queue_length[13] + lane_queue_length[14]; // NEMA Phase 2
	individual_phase_queue_length[2] = lane_queue_length[11]; // NEMA Phase 3
	individual_phase_queue_length[3] = lane_queue_length[0] +
lane_queue_length[1] + lane_queue_length[2]; // NEMA Phase 4
	individual_phase_queue_length[4] = lane_queue_length[15]; // NEMA Phase 5
```

```
	individual_phase_queue_length[5] = lane_queue_length[4] +
lane_queue_length[5] + lane_queue_length[6]; // NEMA Phase 6
	individual_phase_queue_length[6] = lane_queue_length[3]; // NEMA Phase 7
	individual_phase_queue_length[7] = lane_queue_length[8] +
lane_queue_length[9] + lane_queue_length[10]; // NEMA Phase 8

	output = individual_phase_queue_length;
	return individual_phase_queue_length; //output;
}

int Find_Phase_Queue_Length(int intersectionIndex, char *phaseString, VEHICLE*
out_temp_queueTailIDs[16])
{
	//This function finds the total queue length for a given phase, summing up
all lanes and specified phases
	//NOTE: This function simply counts all vehicles on the lane-link closest
to intersection regardless of range or speed
	int intIndex = intersectionIndex;
	int total_queue_length = 0;
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve

	VEHICLE *tempVp; //used in the process of finding the vehicle pointer of
the correct vehicle tail
	int temp_lane_queue_length = 0; //used in the case where the actual tail
vehicle of the platoon is not in the correct lane for its intended movement
through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found

	//declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
	LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
	LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
	LINK* EAST_in_local = intersection[intIndex].EAST_in;
	LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local = intersection[intIndex].WEST_out;
	LINK* NORTH_out_local = intersection[intIndex].NORTH_out;
	LINK* EAST_out_local = intersection[intIndex].EAST_out;
	LINK* SOUTH_out_local = intersection[intIndex].SOUTH_out;

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
```

```
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
		lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);
		total_queue_length += lane_queue_length[0] + lane_queue_length[1] +
lane_queue_length[2];

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[0] = GetQueueTailVp(NORTH_in_local, 1,
lane_queue_length[0], WEST_out_local, SOUTH_out_local);//(LINK *incomingLink, int
incomingLane, int lane_queue_length, LINK *permittedNextLink1, LINK
*permittedNextLink2);

		//out_temp_queueTailIDs[1] = qpg_LNK_vehicleTail(NORTH_in_local, 2);
		out_temp_queueTailIDs[1] = GetQueueTailVp(NORTH_in_local, 2,
lane_queue_length[1], SOUTH_out_local, NULL);
		//out_temp_queueTailIDs[2] = qpg_LNK_vehicleTail(NORTH_in_local, 3);
		out_temp_queueTailIDs[2] = GetQueueTailVp(NORTH_in_local, 3,
lane_queue_length[2], SOUTH_out_local, NULL);
	}
	if (phaseString[2] == 'G')
	{
		lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);
		total_queue_length += lane_queue_length[3];

		out_temp_queueTailIDs[3] = GetQueueTailVp(NORTH_in_local, 4,
lane_queue_length[3], EAST_out_local, NULL);//qpg_LNK_vehicleTail(NORTH_in_local,
4);
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
		lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
		lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);
		total_queue_length += lane_queue_length[4] + lane_queue_length[5] +
lane_queue_length[6];

		out_temp_queueTailIDs[4] = GetQueueTailVp(EAST_in_local, 1,
lane_queue_length[4], NORTH_out_local,
WEST_out_local);//qpg_LNK_vehicleTail(EAST_in_local, 1);
		out_temp_queueTailIDs[5] = GetQueueTailVp(EAST_in_local, 2,
lane_queue_length[5], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
2);
		out_temp_queueTailIDs[6] = GetQueueTailVp(EAST_in_local, 3,
lane_queue_length[6], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
3);
	}
```

```
	if (phaseString[5] == 'G')
	{
		lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);
		total_queue_length += lane_queue_length[7];

		out_temp_queueTailIDs[7] = GetQueueTailVp(EAST_in_local, 4,
lane_queue_length[7], SOUTH_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
4);
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
		lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
		lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);
		total_queue_length += lane_queue_length[8] + lane_queue_length[9] +
lane_queue_length[10];

		out_temp_queueTailIDs[8] = GetQueueTailVp(SOUTH_in_local, 1,
lane_queue_length[8], EAST_out_local,
NORTH_out_local);//qpg_LNK_vehicleTail(SOUTH_in_local, 1);
		out_temp_queueTailIDs[9] = GetQueueTailVp(SOUTH_in_local, 2,
lane_queue_length[9], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
2);
		out_temp_queueTailIDs[10] = GetQueueTailVp(SOUTH_in_local, 3,
lane_queue_length[10], NORTH_out_local,
NULL);//qpg_LNK_vehicleTail(SOUTH_in_local, 3);
	}
	if (phaseString[8] == 'G')
	{
		lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);
		total_queue_length += lane_queue_length[11];

		out_temp_queueTailIDs[11] = GetQueueTailVp(SOUTH_in_local, 4,
lane_queue_length[11], WEST_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
4);
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
		lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
		lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);
		total_queue_length += lane_queue_length[12] + lane_queue_length[13]
+ lane_queue_length[14];

		out_temp_queueTailIDs[12] = GetQueueTailVp(WEST_in_local, 1,
lane_queue_length[12], SOUTH_out_local,
EAST_out_local);//qpg_LNK_vehicleTail(WEST_in_local, 1);
```

```
		out_temp_queueTailIDs[13] = GetQueueTailVp(WEST_in_local, 2,
lane_queue_length[13], EAST_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
2);

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[14] = GetQueueTailVp(WEST_in_local, 3,
lane_queue_length[14], EAST_out_local, NULL);
	}
	if (phaseString[11] == 'G')
	{
		lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);
		total_queue_length += lane_queue_length[15];

		out_temp_queueTailIDs[15] = GetQueueTailVp(WEST_in_local, 4,
lane_queue_length[15], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
4);
	}

	return total_queue_length;
}

int Find_Phase_Queue_Length2(int intersectionIndex, char *phaseString, VEHICLE*
out_temp_queueTailIDs[16])
{
	//This function finds the total queue length for a given phase, summing up
all lanes and specified phases ...NOTE: not currently used
	//NOTE: This function simply counts all queued vehicles on the lane-link
closest to intersection that are under a maximum speed threshold (Is this really
worth the effort?)
	int intIndex = intersectionIndex;
	int total_queue_length = 0;
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve

	VEHICLE *tempVp; //used in the process of finding the vehicle pointer of
the correct vehicle tail
	int temp_lane_queue_length = 0; //used in the case where the actual tail
vehicle of the platoon is not in the correct lane for its intended movement
through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found

	//declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
	LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
	LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
	LINK* EAST_in_local = intersection[intIndex].EAST_in;
	LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local = intersection[intIndex].WEST_out;
	LINK* NORTH_out_local = intersection[intIndex].NORTH_out;
	LINK* EAST_out_local = intersection[intIndex].EAST_out;
```

```
	LINK* SOUTH_out_local = intersection[intIndex].SOUTH_out;

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
		lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);
		total_queue_length += lane_queue_length[0] + lane_queue_length[1] +
lane_queue_length[2];

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[0] = GetQueueTailVp(NORTH_in_local, 1,
lane_queue_length[0], WEST_out_local, SOUTH_out_local);//(LINK *incomingLink, int
incomingLane, int lane_queue_length, LINK *permittedNextLink1, LINK
*permittedNextLink2);
		out_temp_queueTailIDs[1] = GetQueueTailVp(NORTH_in_local, 2,
lane_queue_length[1], SOUTH_out_local, NULL);
		out_temp_queueTailIDs[2] = GetQueueTailVp(NORTH_in_local, 3,
lane_queue_length[2], SOUTH_out_local, NULL);
	}
	if (phaseString[2] == 'G')
	{
		lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);
		total_queue_length += lane_queue_length[3];

		out_temp_queueTailIDs[3] = GetQueueTailVp(NORTH_in_local, 4,
lane_queue_length[3], EAST_out_local, NULL);//qpg_LNK_vehicleTail(NORTH_in_local,
4);
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
		lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
		lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);
		total_queue_length += lane_queue_length[4] + lane_queue_length[5] +
lane_queue_length[6];
```

```
		out_temp_queueTailIDs[4] = GetQueueTailVp(EAST_in_local, 1,
lane_queue_length[4], NORTH_out_local,
WEST_out_local);//qpg_LNK_vehicleTail(EAST_in_local, 1);
		out_temp_queueTailIDs[5] = GetQueueTailVp(EAST_in_local, 2,
lane_queue_length[5], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
2);
		out_temp_queueTailIDs[6] = GetQueueTailVp(EAST_in_local, 3,
lane_queue_length[6], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
3);
	}
	if (phaseString[5] == 'G')
	{
		lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);
		total_queue_length += lane_queue_length[7];

		out_temp_queueTailIDs[7] = GetQueueTailVp(EAST_in_local, 4,
lane_queue_length[7], SOUTH_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
4);
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
		lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
		lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);
		total_queue_length += lane_queue_length[8] + lane_queue_length[9] +
lane_queue_length[10];

		out_temp_queueTailIDs[8] = GetQueueTailVp(SOUTH_in_local, 1,
lane_queue_length[8], EAST_out_local,
NORTH_out_local);//qpg_LNK_vehicleTail(SOUTH_in_local, 1);
		out_temp_queueTailIDs[9] = GetQueueTailVp(SOUTH_in_local, 2,
lane_queue_length[9], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
2);
		out_temp_queueTailIDs[10] = GetQueueTailVp(SOUTH_in_local, 3,
lane_queue_length[10], NORTH_out_local,
NULL);//qpg_LNK_vehicleTail(SOUTH_in_local, 3);
	}
	if (phaseString[8] == 'G')
	{
		lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);
		total_queue_length += lane_queue_length[11];

		out_temp_queueTailIDs[11] = GetQueueTailVp(SOUTH_in_local, 4,
lane_queue_length[11], WEST_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
4);
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
```

```
		lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
		lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
		lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);
		total_queue_length += lane_queue_length[12] + lane_queue_length[13]
+ lane_queue_length[14];

		out_temp_queueTailIDs[12] = GetQueueTailVp(WEST_in_local, 1,
lane_queue_length[12], SOUTH_out_local,
EAST_out_local);//qpg_LNK_vehicleTail(WEST_in_local, 1);
		out_temp_queueTailIDs[13] = GetQueueTailVp(WEST_in_local, 2,
lane_queue_length[13], EAST_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
2);

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[14] = GetQueueTailVp(WEST_in_local, 3,
lane_queue_length[14], EAST_out_local, NULL);
	}
	if (phaseString[11] == 'G')
	{
		lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);
		total_queue_length += lane_queue_length[15];

		out_temp_queueTailIDs[15] = GetQueueTailVp(WEST_in_local, 4,
lane_queue_length[15], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
4);
	}

	return total_queue_length;
}

float GetLaneCurrentDelay(LINK* link, int laneID)
{
	float current_lane_delay = 0.0;
	int vehicle_count_on_lane = qpg_LNK_vehicles(link, laneID); //0; //input to
qpg_LNK_vehicleList
	int i = 0;
	VEHICLE_DATA *myVeh; //= (VEHICLE_DATA*)qpg_VHC_userdata(Vp);
	VEHICLE* vehicle[100]; //make sure that this is sufficiently long to
accomodate maximum queue length based on link length

	qpg_LNK_vehicleList(link, laneID, vehicle_count_on_lane, vehicle); //note:
vehicle must be on the specified link, for its current delay to be aggregated

	for (i = 0; i < vehicle_count_on_lane; i++)
	{
		myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(vehicle[i]);

		current_lane_delay += myVeh->currentNetworkDelay; //this may not be
appropriate for multiple intersections
	}
```

```
	//qps_GUI_printf("Link %s, Lane %d, has %d vehicles with current lane delay
of: %f \n", qpg_LNK_name(link), laneID, vehicle_count_on_lane,
current_lane_delay); //confirmed this function is working

	return current_lane_delay;
}

float Find_Phase_Current_Delay(int intersectionIndex, char *phaseString, VEHICLE*
out_temp_queueTailIDs[16])
{
	//This function finds the total current delay for a given phase, summing up
all lanes and specified phases (only vehicles on those lanes count towards the
current delay for the phase)
	//NOTE: This function simply counts all vehicles on the lane-link closest
to intersection regardless of range or speed
	int intIndex = intersectionIndex;
	int total_queue_length = 0;
	float total_current_delay = 0.0;
	int lane_queue_length[16] = {0}; //array length should be long enough to
accomodate the maximum number of lanes that any phase may serve
	float lane_current_delay[16] = {0.0}; //array length should be long enough
to accomodate the maximum number of lanes that any phase may serve

	VEHICLE *tempVp; //used in the process of finding the vehicle pointer of
the correct vehicle tail
	int temp_lane_queue_length = 0; //used in the case where the actual tail
vehicle of the platoon is not in the correct lane for its intended movement
through the intersection
	Bool FOUND = PFALSE; //true if true queue tail (first vehicle from the rear
of the queue with a correct destination for the given lane) is found

	//declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
	LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
	LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
	LINK* EAST_in_local = intersection[intIndex].EAST_in;
	LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

	//outbound links are needed to correctly find queue tail vehicle
	LINK* WEST_out_local = intersection[intIndex].WEST_out;
	LINK* NORTH_out_local = intersection[intIndex].NORTH_out;
	LINK* EAST_out_local = intersection[intIndex].EAST_out;
	LINK* SOUTH_out_local = intersection[intIndex].SOUTH_out;

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase
```

```
	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[0] = qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_queue_length[1] = qpg_LNK_vehicles(NORTH_in_local, 2);
		lane_queue_length[2] = qpg_LNK_vehicles(NORTH_in_local, 3);
		total_queue_length += lane_queue_length[0] + lane_queue_length[1] +
lane_queue_length[2];

		lane_current_delay[0] = GetLaneCurrentDelay(NORTH_in_local,
1);//qpg_LNK_vehicles(NORTH_in_local, 1);
		lane_current_delay[1] = GetLaneCurrentDelay(NORTH_in_local, 2);
		lane_current_delay[2] = GetLaneCurrentDelay(NORTH_in_local, 3);

		total_current_delay += lane_current_delay[0] + lane_current_delay[1]
+ lane_current_delay[2];

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[0] = GetQueueTailVp(NORTH_in_local, 1,
lane_queue_length[0], WEST_out_local, SOUTH_out_local);//(LINK *incomingLink, int
incomingLane, int lane_queue_length, LINK *permittedNextLink1, LINK
*permittedNextLink2);
		out_temp_queueTailIDs[1] = GetQueueTailVp(NORTH_in_local, 2,
lane_queue_length[1], SOUTH_out_local, NULL);
		out_temp_queueTailIDs[2] = GetQueueTailVp(NORTH_in_local, 3,
lane_queue_length[2], SOUTH_out_local, NULL);
	}
	if (phaseString[2] == 'G')
	{
		lane_queue_length[3] = qpg_LNK_vehicles(NORTH_in_local, 4);
		total_queue_length += lane_queue_length[3];

		lane_current_delay[3] = GetLaneCurrentDelay(NORTH_in_local, 4);

		total_current_delay += lane_current_delay[3];

		out_temp_queueTailIDs[3] = GetQueueTailVp(NORTH_in_local, 4,
lane_queue_length[3], EAST_out_local, NULL);//qpg_LNK_vehicleTail(NORTH_in_local,
4);
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[4] = qpg_LNK_vehicles(EAST_in_local, 1);
		lane_queue_length[5] = qpg_LNK_vehicles(EAST_in_local, 2);
		lane_queue_length[6] = qpg_LNK_vehicles(EAST_in_local, 3);
		total_queue_length += lane_queue_length[4] + lane_queue_length[5] +
lane_queue_length[6];
```

```
		lane_current_delay[4] = GetLaneCurrentDelay(EAST_in_local, 1);
		lane_current_delay[5] = GetLaneCurrentDelay(EAST_in_local, 2);
		lane_current_delay[6] = GetLaneCurrentDelay(EAST_in_local, 3);

		total_current_delay += lane_current_delay[4] + lane_current_delay[5]
+ lane_current_delay[6];

		out_temp_queueTailIDs[4] = GetQueueTailVp(EAST_in_local, 1,
lane_queue_length[4], NORTH_out_local,
WEST_out_local);//qpg_LNK_vehicleTail(EAST_in_local, 1);
		out_temp_queueTailIDs[5] = GetQueueTailVp(EAST_in_local, 2,
lane_queue_length[5], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
2);
		out_temp_queueTailIDs[6] = GetQueueTailVp(EAST_in_local, 3,
lane_queue_length[6], WEST_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
3);
	}
	if (phaseString[5] == 'G')
	{
		lane_queue_length[7] = qpg_LNK_vehicles(EAST_in_local, 4);
		total_queue_length += lane_queue_length[7];

		lane_current_delay[7] = GetLaneCurrentDelay(EAST_in_local, 4);

		total_current_delay += lane_current_delay[7];

		out_temp_queueTailIDs[7] = GetQueueTailVp(EAST_in_local, 4,
lane_queue_length[7], SOUTH_out_local, NULL);//qpg_LNK_vehicleTail(EAST_in_local,
4);
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[8] = qpg_LNK_vehicles(SOUTH_in_local, 1);
		lane_queue_length[9] = qpg_LNK_vehicles(SOUTH_in_local, 2);
		lane_queue_length[10] = qpg_LNK_vehicles(SOUTH_in_local, 3);
		total_queue_length += lane_queue_length[8] + lane_queue_length[9] +
lane_queue_length[10];

		lane_current_delay[8] = GetLaneCurrentDelay(SOUTH_in_local, 1);
		lane_current_delay[9] = GetLaneCurrentDelay(SOUTH_in_local, 2);
		lane_current_delay[10] = GetLaneCurrentDelay(SOUTH_in_local, 3);

		total_current_delay += lane_current_delay[8] + lane_current_delay[9]
+ lane_current_delay[10];

		out_temp_queueTailIDs[8] = GetQueueTailVp(SOUTH_in_local, 1,
lane_queue_length[8], EAST_out_local,
NORTH_out_local);//qpg_LNK_vehicleTail(SOUTH_in_local, 1);
```

```
		out_temp_queueTailIDs[9] = GetQueueTailVp(SOUTH_in_local, 2,
lane_queue_length[9], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
2);
		out_temp_queueTailIDs[10] = GetQueueTailVp(SOUTH_in_local, 3,
lane_queue_length[10], NORTH_out_local,
NULL);//qpg_LNK_vehicleTail(SOUTH_in_local, 3);
	}
	if (phaseString[8] == 'G')
	{
		lane_queue_length[11] = qpg_LNK_vehicles(SOUTH_in_local, 4);
		total_queue_length += lane_queue_length[11];

		lane_current_delay[11] = GetLaneCurrentDelay(SOUTH_in_local, 4);

		total_current_delay += lane_current_delay[11];

		out_temp_queueTailIDs[11] = GetQueueTailVp(SOUTH_in_local, 4,
lane_queue_length[11], WEST_out_local, NULL);//qpg_LNK_vehicleTail(SOUTH_in_local,
4);
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		lane_queue_length[12] = qpg_LNK_vehicles(WEST_in_local, 1);
		lane_queue_length[13] = qpg_LNK_vehicles(WEST_in_local, 2);
		lane_queue_length[14] = qpg_LNK_vehicles(WEST_in_local, 3);
		total_queue_length += lane_queue_length[12] + lane_queue_length[13]
+ lane_queue_length[14];

		lane_current_delay[12] = GetLaneCurrentDelay(WEST_in_local, 1);
		lane_current_delay[13] = GetLaneCurrentDelay(WEST_in_local, 2);
		lane_current_delay[14] = GetLaneCurrentDelay(WEST_in_local, 3);

		total_current_delay += lane_current_delay[12] +
lane_current_delay[13] + lane_current_delay[14];

		out_temp_queueTailIDs[12] = GetQueueTailVp(WEST_in_local, 1,
lane_queue_length[12], SOUTH_out_local,
EAST_out_local);//qpg_LNK_vehicleTail(WEST_in_local, 1);
		out_temp_queueTailIDs[13] = GetQueueTailVp(WEST_in_local, 2,
lane_queue_length[13], EAST_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
2);

		//need to account for situation when tail changes lanes
		// METHOD 2: generic code within separate function
		out_temp_queueTailIDs[14] = GetQueueTailVp(WEST_in_local, 3,
lane_queue_length[14], EAST_out_local, NULL);

	}
	if (phaseString[11] == 'G')
	{
		lane_queue_length[15] = qpg_LNK_vehicles(WEST_in_local, 4);
```

```
		total_queue_length += lane_queue_length[15];

		lane_current_delay[15] = GetLaneCurrentDelay(WEST_in_local, 4);

		total_current_delay += lane_current_delay[15];

		out_temp_queueTailIDs[15] = GetQueueTailVp(WEST_in_local, 4,
lane_queue_length[15], NORTH_out_local, NULL);//qpg_LNK_vehicleTail(WEST_in_local,
4);
	}

	//qps_GUI_printf("total current delay = %f \n", total_current_delay);
//confirmed this function is working

	return total_current_delay;
}

float Find_Phase_Current_Delay0(int intersectionIndex, char *phaseString)
{
	//This function finds ONLY the total current delay for a given phase,
summing up all lanes and specified phases (only vehicles on those lanes count
towards the current delay for the phase)
	//NOTE: this function does not find the queue tails, and is intended only
for calculating the current delay
	int intIndex = intersectionIndex;
	float total_current_delay = 0.0;
	float lane_current_delay[16] = {0.0}; //array length should be long enough
to accomodate the maximum number of lanes that any phase may serve

	//declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
	LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
	LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
	LINK* EAST_in_local = intersection[intIndex].EAST_in;
	LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

	//currentQueueIDList = [] #initialize list as empty at the beginning of
each function call

	//need list of vehicles within communication radius
	// for starters, assume queues are less than 600 ft, then only the incoming
links connected with the isolated intersection need to be considered
	//		also, simply start by counting all the vehicles on the
incoming links (i.e there is no maximum speed constraint on queue definition)
	//	1) calculate maximum combined queue length: sum over all lanes of
the prospective phase to see how many vehicles would be served by activating this
phase

	//NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
	if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
```

```
                lane_current_delay[0] = GetLaneCurrentDelay(NORTH_in_local,
1);//qpg_LNK_vehicles(NORTH_in_local, 1);
                lane_current_delay[1] = GetLaneCurrentDelay(NORTH_in_local, 2);
                lane_current_delay[2] = GetLaneCurrentDelay(NORTH_in_local, 3);

                total_current_delay += lane_current_delay[0] + lane_current_delay[1]
+ lane_current_delay[2];
        }
        if (phaseString[2] == 'G')
        {
                lane_current_delay[3] = GetLaneCurrentDelay(NORTH_in_local, 4);

                total_current_delay += lane_current_delay[3];
        }

        if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
        {
                lane_current_delay[4] = GetLaneCurrentDelay(EAST_in_local, 1);
                lane_current_delay[5] = GetLaneCurrentDelay(EAST_in_local, 2);
                lane_current_delay[6] = GetLaneCurrentDelay(EAST_in_local, 3);

                total_current_delay += lane_current_delay[4] + lane_current_delay[5]
+ lane_current_delay[6];
        }
        if (phaseString[5] == 'G')
        {
                lane_current_delay[7] = GetLaneCurrentDelay(EAST_in_local, 4);

                total_current_delay += lane_current_delay[7];
        }

        if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
        {
                lane_current_delay[8] = GetLaneCurrentDelay(SOUTH_in_local, 1);
                lane_current_delay[9] = GetLaneCurrentDelay(SOUTH_in_local, 2);
                lane_current_delay[10] = GetLaneCurrentDelay(SOUTH_in_local, 3);

                total_current_delay += lane_current_delay[8] + lane_current_delay[9]
+ lane_current_delay[10];
        }
        if (phaseString[8] == 'G')
        {
                lane_current_delay[11] = GetLaneCurrentDelay(SOUTH_in_local, 4);

                total_current_delay += lane_current_delay[11];
        }

        if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
```

```
	{
		lane_current_delay[12] = GetLaneCurrentDelay(WEST_in_local, 1);
		lane_current_delay[13] = GetLaneCurrentDelay(WEST_in_local, 2);
		lane_current_delay[14] = GetLaneCurrentDelay(WEST_in_local, 3);

		total_current_delay += lane_current_delay[12] +
lane_current_delay[13] + lane_current_delay[14];
	}
	if (phaseString[11] == 'G')
	{
		lane_current_delay[15] = GetLaneCurrentDelay(WEST_in_local, 4);

		total_current_delay += lane_current_delay[15];
	}

	//qps_GUI_printf("total current delay = %f \n", total_current_delay);
//confirmed this function is working

	return total_current_delay;
}

void OPTIMIZE5(char *intersectionID)
{
	//Traffic Signal Light Optimizer: Dynamic Programming Method, Max Queue
Length Optimizer: Full-Queue Discharge Version
	int i;
	int j;
	int k;
	char *currentStatus; //check if this works
    int max_queue_length = 0; //default initialization
    int BESTPHASE_INDEX = 48; //default phase is ALLRED
	int nextPhaseIndex;
	char *TESTPHASE;
	char *TESTPHASE2;
	int queue_length;
	int currentQueueIDList_out[100]; //not used
	int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8 cases, (all
green phases), NOTE: this is used in the (currentStatus == "G") branch
	int testCasesFromAllRed[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //in case
definition above causes problems, just start with constant matrix entries
	VEHICLE* temp_queueTailIDs[16];
	int intIndex = 0; //intersection index, NOTE: this is intended for a single
corridor of intersections, use 2 indexes for 2D matrix for irongrid network

	//retrieve intersection index based on supplied intersection ID
	intIndex = index_from_ID(intersectionID);

    //add condition for iterating through the necessary test cases
    currentStatus = ALLPHASESSTATUS[intersection[intIndex].currentPhaseIndex];

    if ((currentStatus == "G") && (intersection[intIndex].queuesDischarged ==
PFALSE) && (intersection[intIndex].currentPhaseDuration < maximumGreenDuration))
//(len(QueueIDList) > 0)) //(queuesDischarged == False): NOTE: can replace
len(QueueIDList) with a QueueLength variable, or use queuesDischarged boolean
```

```
        {
         intersection[intIndex].currentPhaseDuration += 1; //extend green until
current phases' queues are fully discharged

              //test section
              if (intersection[intIndex].currentPhaseDuration >
intersection[intIndex].maxGreenReached)
              {
                     intersection[intIndex].maxGreenReached =
intersection[intIndex].currentPhaseDuration;
              }
        }
    else if (currentStatus == "G") //CONDITION 2) described in IMA2(); check
transition to all 8 greens      //queuesDischarged == True
        {
         for (i = 0; i < 8; i++)
              {
            TESTPHASE = ALLPHASES[testCases[i]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(intIndex, TESTPHASE,
&temp_queueTailIDs); //currentQueueIDList_out?
            if (queue_length > max_queue_length) //what about ties?
                     {
                max_queue_length = queue_length;
                BESTPHASE_INDEX = testCases[i]; //case index is only correct for
extending given green
                           for (k = 0; k < 16; k++)
                           {
                                  intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                           }
                     }
              }

         //4 possible cases
         // 1) Current green is extended 1 second (1/8)
         // 2) First movement stays green during transition to next green    (e.g.
1/5 to 1/6 with 1 continuing) (1/8)
         // 3) Second movement stays green during transition to next green (e.g.
1/5 to 2/5 with 5 continuing) (1/8)
         // 4) Both movements go to yellow before going to all red (5/8)
         if (NEXTPHASES[intersection[intIndex].currentPhaseIndex][0] ==
BESTPHASE_INDEX) //1)
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]; //actually already set
above
              }
         else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][2]][0]
][0] == BESTPHASE_INDEX) //2) check if first movement stays green
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][2];
```

```
            }
        else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][3]][0]
][0] == BESTPHASE_INDEX) //3) check if second movement stays green
            {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][3];
            }
        else //4) must transition to all red on route to next green
            {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][1];
            }

        if (BESTPHASE_INDEX ==
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]) //check if original green
phase is extended
            {
            intersection[intIndex].currentPhaseDuration += 1;
            }
        else //phase has switched to a yellow phase
            {
            nextPhaseIndex = BESTPHASE_INDEX;
                    SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
//light changes to yellow
            intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
            intersection[intIndex].currentPhaseDuration = 3; //default duration
for a yellow phase
            intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
            }
       }
    else if (currentStatus == "R") //CONDITION 1) described in IMA2()
       {
        for (j = 0; j < 8; j++)
            {
            TESTPHASE2 = ALLPHASES[testCasesFromAllRed[j]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(intIndex, TESTPHASE2,
&temp_queueTailIDs); //, currentQueueIDList_out?
            if (queue_length > max_queue_length) //what about ties?
                    {
                max_queue_length = queue_length;
                BESTPHASE_INDEX = testCasesFromAllRed[j]; //case index is only
correct for extending given green

                            //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                            for (k = 0; k < 16; k++)
                            {
                                    intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                            }
                    }
```

```
		}

		//hold the all red if there are no vehicles in range (this provides
flexibility to change the signal quickly to an appropriate green phase)
		if (max_queue_length == 0) //then no vehicles were found in range
(NOTE: remove this all red hold policy in the case of unequipped vehicles)
		{
			//ALL-RED Policy: extend all-red phase until a vehicle is
detected in range (provides flexibility in serving incoming vehicles quickly)
			intersection[intIndex].currentPhaseDuration += 1; //look for
vehicles again, after 1 second
		}
		else //at least 1 vehicle found, light must change to green
		{
			nextPhaseIndex = BESTPHASE_INDEX;
			SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
		//light changes to green
			intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
			intersection[intIndex].currentPhaseDuration =
minimumGreenDuration; //default minimum green is 8 seconds
			intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
			intersection[intIndex].queuesDischarged = PFALSE; //check if
this reset works
		}
	}
}

void OPTIMIZE6(char *intersectionID)
{
	//Traffic Signal Light Optimizer: Dynamic Programming Method, Max Queue
Length Optimizer: Full-Queue Discharge Version, with Dual Phase Quadratic Aging
Factors
	int i;
	int j;
	int k;
	char *currentStatus; //check if this works
    int max_queue_length = 0; //default initialization
    int BESTPHASE_INDEX = 48; //default phase is ALLRED
	int nextPhaseIndex;
	char *TESTPHASE;
	char *TESTPHASE2;
	int queue_length;
	float queue_lengthXagingFactor = 0.0;
	float max_queue_lengthXagingFactor = 0.0;
	int currentQueueIDList_out[100];
	int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8 cases, (all
green phases), NOTE: this is used in the (currentStatus == "G") branch
	int testCasesFromAllRed[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //in case
definition above causes problems, just start with constant matrix entries
	VEHICLE* temp_queueTailIDs[16];
	int intIndex = 0; //intersection index, NOTE: this is intended for a single
corridor of intersections, use 2 indexes for 2D matrix for irongrid network
```

```
        //retrieve intersection index based on supplied intersection ID
        intIndex = index_from_ID(intersectionID);

    //add condition for iterating through the necessary test cases
    currentStatus = ALLPHASESSTATUS[intersection[intIndex].currentPhaseIndex];

    if ((currentStatus == "G") && (intersection[intIndex].queuesDischarged ==
PFALSE) && (intersection[intIndex].currentPhaseDuration < maximumGreenDuration))
//(len(QueueIDList) > 0)) //(queuesDischarged == False): NOTE: can replace
len(QueueIDList) with a QueueLength variable, or use queuesDischarged boolean
        {
         intersection[intIndex].currentPhaseDuration += 1; //extend green until
current phases' queues are fully discharged

              //test section
              if (intersection[intIndex].currentPhaseDuration >
intersection[intIndex].maxGreenReached)
              {
                     intersection[intIndex].maxGreenReached =
intersection[intIndex].currentPhaseDuration;
              }
        }
    else if (currentStatus == "G") //CONDITION 2) described in IMA2(); check
transition to all 8 greens      //queuesDischarged == True
        {

         for (i = 0; i < 8; i++)
              {
            TESTPHASE = ALLPHASES[testCases[i]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(intIndex, TESTPHASE,
&temp_queueTailIDs); //currentQueueIDList_out?
                     queue_lengthXagingFactor = queue_length *
intersection[intIndex].agingFactor[i];

            if (queue_lengthXagingFactor > max_queue_lengthXagingFactor) //what
about ties?
                     {
                max_queue_lengthXagingFactor = queue_lengthXagingFactor;
                BESTPHASE_INDEX = testCases[i]; //case index is only correct for
extending given green

                             //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                             for (k = 0; k < 16; k++)
                             {
                                    intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                             }
                     }
              }

         //4 possible cases
         // 1) Current green is extended 1 second (1/8)
```

```
        // 2) First movement stays green during transition to next green    (e.g.
1/5 to 1/6 with 1 continuing) (1/8)
        // 3) Second movement stays green during transition to next green (e.g.
1/5 to 2/5 with 5 continuing) (1/8)
        // 4) Both movements go to yellow before going to all red (5/8)
        if (NEXTPHASES[intersection[intIndex].currentPhaseIndex][0] ==
BESTPHASE_INDEX) //1)
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]; //actually already set
above
              }
        else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][2]][0]
][0] == BESTPHASE_INDEX) //2) check if first movement stays green
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][2];
              }
        else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][3]][0]
][0] == BESTPHASE_INDEX) //3) check if second movement stays green
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][3];
              }
        else //4) must transition to all red on route to next green
              {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][1];
              }

        if (BESTPHASE_INDEX ==
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]) //check if original green
phase is extended
              {
            intersection[intIndex].currentPhaseDuration += 1;
              }
        else //phase has switched to a yellow phase
              {
            nextPhaseIndex = BESTPHASE_INDEX;
                    SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
//light changes to yellow
            intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
            intersection[intIndex].currentPhaseDuration = 3; //default duration
for a yellow phase
            intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
              }
       }
    else if (currentStatus == "R") //CONDITION 1) described in IMA2()
       {
        for (j = 0; j < 8; j++)
              {
```

```
            TESTPHASE2 = ALLPHASES[testCasesFromAllRed[j]];
            //evaluate benefit of decision: calculate maximum queue length
            queue_length = Find_Phase_Queue_Length(intIndex, TESTPHASE2,
&temp_queueTailIDs); //, currentQueueIDList_out?
                     queue_lengthXagingFactor = queue_length *
intersection[intIndex].agingFactor[j];
                     if (queue_lengthXagingFactor > max_queue_lengthXagingFactor)
//what about ties?
                     {
                max_queue_lengthXagingFactor = queue_lengthXagingFactor;
                BESTPHASE_INDEX = testCasesFromAllRed[j]; //case index is only
correct for extending given green

                           for (k = 0; k < 16; k++)
                           {
                                  intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                           }
                     }
              }

        //light must change to green
        nextPhaseIndex = BESTPHASE_INDEX;
        SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
              //light changes to green
        intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
        intersection[intIndex].currentPhaseDuration = minimumGreenDuration;
//default minimum green is 8 seconds
        intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
              intersection[intIndex].queuesDischarged = PFALSE; //check if this
reset works
       }
}

void OPTIMIZE7(char *intersectionID)
{
       //Traffic Signal Light Optimizer: Dynamic Programming Method, Current
Network Delay Version, (with full queue discharge)
       int i;
       int j;
       int k;
       char *currentStatus; //check if this works
    int max_queue_length = 0; //default initialization
    int BESTPHASE_INDEX = 48; //default phase is ALLRED
       int nextPhaseIndex;
       char *TESTPHASE;
       char *TESTPHASE2;
       int queue_length;
       float queue_lengthXagingFactor = 0.0;
       float max_queue_lengthXagingFactor = 0.0;
       float current_delay = 0.0;
       float max_current_delay = 0.0;
       int currentQueueIDList_out[100];
```

```
        int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8 cases, (all
green phases), NOTE: this is used in the (currentStatus == "G") branch
        int testCasesFromAllRed[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //in case
definition above causes problems, just start with constant matrix entries
        VEHICLE* temp_queueTailIDs[16];
        int intIndex = 0; //intersection index, NOTE: this is intended for a single
corridor of intersections, use 2 indexes for 2D matrix for irongrid network

        //retrieve intersection index based on supplied intersection ID
        intIndex = index_from_ID(intersectionID);

    //add condition for iterating through the necessary test cases
    currentStatus = ALLPHASESSTATUS[intersection[intIndex].currentPhaseIndex];

    if ((currentStatus == "G") && (intersection[intIndex].queuesDischarged ==
PFALSE) && (intersection[intIndex].currentPhaseDuration < maximumGreenDuration))
//(len(QueueIDList) > 0)) //(queuesDischarged == False): NOTE: can replace
len(QueueIDList) with a QueueLength variable, or use queuesDischarged boolean
        {
         intersection[intIndex].currentPhaseDuration += 1; //extend green until
current phases' queues are fully discharged

              //test section
              if (intersection[intIndex].currentPhaseDuration >
intersection[intIndex].maxGreenReached)
              {
                     intersection[intIndex].maxGreenReached =
intersection[intIndex].currentPhaseDuration;
              }
        }
    else if (currentStatus == "G") //CONDITION 2) described in IMA2(); check
transition to all 8 greens      //queuesDischarged == True
        {
              //NOTE: can add case later, where if max green is reached or is
within 8 seconds of being reached, then only 7 cases should be considered

              //int testCases[8] = {0, 6, 12, 18, 24, 30, 36, 42}; //evaluate 8
cases, (all green phases), NOTE: this is declared above
         for (i = 0; i < 8; i++)
              {
            TESTPHASE = ALLPHASES[testCases[i]];
            //evaluate benefit of decision: calculate maximum current delay
                     current_delay = Find_Phase_Current_Delay(intIndex, TESTPHASE,
&temp_queueTailIDs);

            if (current_delay > max_current_delay) //what about ties?
                     {
                max_current_delay = current_delay;
                BESTPHASE_INDEX = testCases[i]; //case index is only correct for
extending given green

                           //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                           for (k = 0; k < 16; k++)
```

```
                            {
                                    intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                            }
                        }
                }

        //4 possible cases
        // 1) Current green is extended 1 second (1/8)
        // 2) First movement stays green during transition to next green    (e.g.
1/5 to 1/6 with 1 continuing) (1/8)
        // 3) Second movement stays green during transition to next green (e.g.
1/5 to 2/5 with 5 continuing) (1/8)
        // 4) Both movements go to yellow before going to all red (5/8)
        if (NEXTPHASES[intersection[intIndex].currentPhaseIndex][0] ==
BESTPHASE_INDEX) //1)
                {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]; //actually already set
above
                }
        else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][2]][0]
][0] == BESTPHASE_INDEX) //2) check if first movement stays green
                {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][2];
                }
        else if
(NEXTPHASES[NEXTPHASES[NEXTPHASES[intersection[intIndex].currentPhaseIndex][3]][0]
][0] == BESTPHASE_INDEX) //3) check if second movement stays green
                {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][3];
                }
        else //4) must transition to all red on route to next green
                {
            BESTPHASE_INDEX =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][1];
                }

        if (BESTPHASE_INDEX ==
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0]) //check if original green
phase is extended
                {
            intersection[intIndex].currentPhaseDuration += 1;
                }
        else //phase has switched to a yellow phase
                {
            nextPhaseIndex = BESTPHASE_INDEX;
                    SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
//light changes to yellow
            intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
```

```
            intersection[intIndex].currentPhaseDuration = 3; //default duration
for a yellow phase
            intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
              }
       }
    else if (currentStatus == "R") //CONDITION 1) described in IMA2()
       {
        for (j = 0; j < 8; j++)
              {
            TESTPHASE2 = ALLPHASES[testCasesFromAllRed[j]];
            //evaluate benefit of decision: calculate maximum queue length
                     current_delay = Find_Phase_Current_Delay(intIndex, TESTPHASE2,
&temp_queueTailIDs);
                     if (current_delay > max_current_delay) //what about ties?
                     {
                max_current_delay = current_delay;
                BESTPHASE_INDEX = testCasesFromAllRed[j]; //case index is only
correct for extending given green

                           //queueTailIDs = temp_queueTailIDs; //haha, see if this
works
                           for (k = 0; k < 16; k++)
                           {
                                  intersection[intIndex].queueTailIDs[k] =
temp_queueTailIDs[k]; //see if this works
                           }
                     }
              }

        //light must change to green
        nextPhaseIndex = BESTPHASE_INDEX;
        SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
              //light changes to green
        intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
        intersection[intIndex].currentPhaseDuration = minimumGreenDuration;
//default minimum green is 8 seconds
        intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
              intersection[intIndex].queuesDischarged = PFALSE; //check if this
reset works
       }
}

void IMA2(char *intersectionID)
{
       //Intersection Management Agent queries Optimize#() if necessary, //NOTE:
Decentralized framework has one IMA per intersection in network
    //Optimization is only necessary under the following 2 conditions
    //CONDITION 1) An ALLRED phase has just expired, OPTIMIZE#() must decide which
green phase to switch to
    //CONDITION 2) A double green phase is past its min green time of 8 seconds
       char *currentStatus; //check if this works
       float currentTime;
       int nextPhaseIndex;
```

```
       int intIndex = 0; //intersection index, NOTE: this is intended for a single
corridor of intersections, use 2 indexes for 2D matrix for irongrid network

       //retrieve intersection index based on supplied intersection ID
       intIndex = index_from_ID(intersectionID);

    // determine if OPTIMIZE# needs to be called
       currentTime = qpg_CFG_simulationTime(); //time returned by paramics should
have units of seconds

       //qps_GUI_printf("IMA2 entered for intersection %s at time %f, intersection
index = % d \n", intersectionID, currentTime, index_from_ID(intersectionID));

    //check light status as defined in ALLPHASESSTATUS
    currentStatus = ALLPHASESSTATUS[intersection[intIndex].currentPhaseIndex];
    if (currentStatus == "Y") //light remains yellow until changed
       {
        if ((currentTime - intersection[intIndex].phase_start) >=
intersection[intIndex].currentPhaseDuration) //currentPhaseDuration should be 3
seconds here
             {
            //change light to appropriate red
            nextPhaseIndex =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0];
                    SetRedYellowGreenState(intIndex, ALLPHASES[nextPhaseIndex]);
//light changes to red
            intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
            intersection[intIndex].currentPhaseDuration = 1; //all phase
combinations with status == "R" currently have a fixed duration of 1 second
            intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
             }
       }
    else if (currentStatus == "R") //light remains red until changed
       {
        if ((currentTime - intersection[intIndex].phase_start) >=
intersection[intIndex].currentPhaseDuration) //currentPhaseDuration should be 1
second here, (unless the ALL-RED policy is in effect)
             {
            //check if red is all red
            if (intersection[intIndex].currentPhaseIndex == 48) //48 is the index
for ALLRED
                    {
                OPTIMIZE6(intersectionID); //5, 6, 7
                    }
            else //must be a single green phase, therefore, transition to a double
green phase (see FlexibleTrafficLightStateMachine_colored.vsd)
                    {
                //change light to appropriate green
                nextPhaseIndex =
NEXTPHASES[intersection[intIndex].currentPhaseIndex][0];
                            SetRedYellowGreenState(intIndex,
ALLPHASES[nextPhaseIndex]);               //light changes to green
```

```
                intersection[intIndex].phase_start +=
intersection[intIndex].currentPhaseDuration; //reset phase_start
                intersection[intIndex].currentPhaseDuration =
minimumGreenDuration; //reset current phase duration to min green
                intersection[intIndex].currentPhaseIndex = nextPhaseIndex;
                            intersection[intIndex].queuesDischarged = PFALSE;
//check if this reset works
                      }
              }
       }
       else if (currentStatus == "G") //light remains green until changed, should
remain green for at least the duration of min green
       {
        if ((currentTime - intersection[intIndex].phase_start) >=
intersection[intIndex].currentPhaseDuration) //should be at least 8 seconds,
OPTIMIZE//() can extend the duration in increments of 1 second
              {
            OPTIMIZE6(intersectionID);    //5, 6, 7
              }
       }
}

Bool QueuesDischargedCheck(int intersectionIndex, char *phaseString)
{
       int intIndex = intersectionIndex;
       Bool temp_queuesDischarged = PTRUE; //initialization must be 'true'

       //declare pointers (these are the links connected to the junction specified
by the intersectionIndex parameter)
       LINK* WEST_in_local = intersection[intIndex].WEST_in; //check if this works
       LINK* NORTH_in_local = intersection[intIndex].NORTH_in;
       LINK* EAST_in_local = intersection[intIndex].EAST_in;
       LINK* SOUTH_in_local = intersection[intIndex].SOUTH_in;

       //NOTE: can use intersectionID later to add additional intersections
       //NODE* n = qpg_NET_nodeByIndex(intersectionID); //check if this index
works

       //NOTE: can add code later to make this applicable to other intersections
(e.g. different number of lanes, or different lane movements)
       //NOTE: Queue is discharged when the front bumper of the designated tail
vehicle crosses the stop bar
       if ((phaseString[0] == 'G') && (phaseString[1] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
       {
              if ((qpg_VHC_link(intersection[intIndex].queueTailIDs[0]) ==
NORTH_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[1]) ==
NORTH_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[2]) ==
NORTH_in_local))
              {
                      temp_queuesDischarged = PFALSE;
              }
       }
```

```
	if (phaseString[2] == 'G')
	{
		if (qpg_VHC_link(intersection[intIndex].queueTailIDs[3]) ==
NORTH_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[3] == 'G') && (phaseString[4] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(intersection[intIndex].queueTailIDs[4]) ==
EAST_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[5]) ==
EAST_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[6]) ==
EAST_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[5] == 'G')
	{
		if (qpg_VHC_link(intersection[intIndex].queueTailIDs[7]) ==
EAST_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[6] == 'G') && (phaseString[7] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
	{
		if ((qpg_VHC_link(intersection[intIndex].queueTailIDs[8]) ==
SOUTH_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[9]) ==
SOUTH_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[10]) ==
SOUTH_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[8] == 'G')
	{
		if (qpg_VHC_link(intersection[intIndex].queueTailIDs[11]) ==
SOUTH_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	if ((phaseString[9] == 'G') && (phaseString[10] == 'G')) //NOTE: This code
assumes that right turns move with through movements;  //check if single quotes
works
```

```
	{
		if ((qpg_VHC_link(intersection[intIndex].queueTailIDs[12]) ==
WEST_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[13]) ==
WEST_in_local) || (qpg_VHC_link(intersection[intIndex].queueTailIDs[14]) ==
WEST_in_local))
		{
			temp_queuesDischarged = PFALSE;
		}
	}
	if (phaseString[11] == 'G')
	{
		if (qpg_VHC_link(intersection[intIndex].queueTailIDs[15]) ==
WEST_in_local)
		{
			temp_queuesDischarged = PFALSE;
		}
	}

	return temp_queuesDischarged;
}

void UpdateTimeLastServed(float currentTime_in, int intersectionIndex)
{
	int intIndex = intersectionIndex;
	//char *lastPhaseString = ALLPHASES[lastPhaseIndex]; //check if this works
	char *currentPhaseString =
ALLPHASES[intersection[intIndex].currentPhaseIndex];
	int i = 0;

	for (i = 0; i < strlen(currentPhaseString); i++) //length should be 12
	{
		//if ((lastPhaseString[i] == 'G') && (currentPhaseString[i] == 'y'))
//update "time last served" when a green transitions to yellow
		if (currentPhaseString[i] == 'G') //update timeLastServed[] while
the light is green
		{
			//Description: The strings use the following characters: 1) G
= green, 2) y = yellow, 3) R = Right Turn On Red, 4) r = red
			//		The movements are ordered cw, starting from the
North_in right turn movement, or more explicitly the 12 movements are as follows
(NEMA phase equivalent in parantheses):
			//		1. North_in, West_out  (4)
			//		2. North_in, South_out (4)
			//		3. North_in, East_out  (7)
			//		4. East_in, North_out  (6)
			//		5. East_in, West_out   (6)
			//		6. East_in, South_out  (1)
			//		7. South_in, East_out  (8)
			//		8. South_in, North_out (8)
			//		9. South_in, West_out  (3)
			//		10. West_in, South_out  (2)
			//		11. West_in, East_out   (2)
			//		12. West_in, North_out  (5)
```

```
                    if ((i == 0) || (i == 1))
                    {
                            intersection[intIndex].timeLastServed[3] =
currentTime_in; //NEMA phase 4
                    }
                    else if (i == 2)
                    {
                            intersection[intIndex].timeLastServed[6] =
currentTime_in; //NEMA phase 7
                    }
                    else if ((i == 3) || (i == 4))
                    {
                            intersection[intIndex].timeLastServed[5] =
currentTime_in; //NEMA phase 6
                    }
                    else if (i == 5)
                    {
                            intersection[intIndex].timeLastServed[0] =
currentTime_in; //NEMA phase 1
                    }
                    else if ((i == 6) || (i == 7))
                    {
                            intersection[intIndex].timeLastServed[7] =
currentTime_in; //NEMA phase 8
                    }
                    else if (i == 8)
                    {
                            intersection[intIndex].timeLastServed[2] =
currentTime_in; //NEMA phase 3
                    }
                    else if ((i == 9) || (i == 10))
                    {
                            intersection[intIndex].timeLastServed[1] =
currentTime_in; //NEMA phase 2
                    }
                    else if (i == 11)
                    {
                            intersection[intIndex].timeLastServed[4] =
currentTime_in; //NEMA phase 5
                    }
            }
        }

}

void UpdateAgingTime(float currentTime_in, int intersectionIndex)
{
        int intIndex = intersectionIndex;
        int i = 0;

        for (i = 0; i < 8; i++) //8 is number of individual phase movements
        {
                intersection[intIndex].agingTime[i] = currentTime_in -
intersection[intIndex].timeLastServed[i];
```

```
	}
}

float CalculateAgingFactor(float dualPhaseAgingTime)
{
	//aging factor equation is a quadratic fitting three points perfectly (0,
1), (30, 2), and (120, 10)
	//This means that an input of 120 seconds time leads to an aging factor of
10, (the queue lengths are multiplied by a factor of 10)
	return 0.000462963 * dualPhaseAgingTime * dualPhaseAgingTime + 0.0194444 *
dualPhaseAgingTime + 1;
}

void UpdateAgingFactors(int intersectionIndex)
{
	int intIndex = intersectionIndex;
	//aging factor muliplies queue length to modulate fairness, based on
elapsed time since a movement has been served
	//NOTE: aging factors refers to dual phases: 1/5, 1/6, 2/5, 2/6, 3/7, 3/8,
4/7, 4/8
	intersection[intIndex].agingFactor[0] =
CalculateAgingFactor((intersection[intIndex].agingTime[0] +
intersection[intIndex].agingTime[4])/2); //NEMA dual phase 1/5
	intersection[intIndex].agingFactor[1] =
CalculateAgingFactor((intersection[intIndex].agingTime[0] +
intersection[intIndex].agingTime[5])/2); //NEMA dual phase 1/6
	intersection[intIndex].agingFactor[2] =
CalculateAgingFactor((intersection[intIndex].agingTime[1] +
intersection[intIndex].agingTime[4])/2); //NEMA dual phase 2/5
	intersection[intIndex].agingFactor[3] =
CalculateAgingFactor((intersection[intIndex].agingTime[1] +
intersection[intIndex].agingTime[5])/2); //NEMA dual phase 2/6
	intersection[intIndex].agingFactor[4] =
CalculateAgingFactor((intersection[intIndex].agingTime[2] +
intersection[intIndex].agingTime[6])/2); //NEMA dual phase 3/7
	intersection[intIndex].agingFactor[5] =
CalculateAgingFactor((intersection[intIndex].agingTime[2] +
intersection[intIndex].agingTime[7])/2); //NEMA dual phase 3/8
	intersection[intIndex].agingFactor[6] =
CalculateAgingFactor((intersection[intIndex].agingTime[3] +
intersection[intIndex].agingTime[6])/2); //NEMA dual phase 4/7
	intersection[intIndex].agingFactor[7] =
CalculateAgingFactor((intersection[intIndex].agingTime[3] +
intersection[intIndex].agingTime[7])/2); //NEMA dual phase 4/8
}

void UpdateAgingFactors2(int intersectionIndex)
{
	//UpdateAgingFactor2 accounts for the number of vehicles per individual
movement
	//aging factor muliplies queue length to modulate fairness, based on
elapsed time since a movement has been served
	//NOTE: aging factors refers to dual phases: 1/5, 1/6, 2/5, 2/6, 3/7, 3/8,
4/7, 4/8
```

```
	int intIndex = intersectionIndex;
	int individual_phase_queue_lengths[8] = {1};
	int *p;
	int i;
	// 1) First get queue lengths for each individual NEMA phase
	//individual_phase_queue_lengths[0] =
Find_All_Individual_Phase_Queue_Lengths(5); //can update later to account for
multiple intersections, ...check if this works
	p = Find_All_Individual_Phase_Queue_Lengths(intIndex); //can update later
to account for multiple intersections, ...check if this works

	for (i = 0; i < 8; i++)
	{
		individual_phase_queue_lengths[i] = *(p + i);
	}

	//NOTE: the max function is used in the denominator of the calculations
below to avoid dividing by 0 when there are no vehicles on either phase
	intersection[intIndex].agingFactor[0] =
CalculateAgingFactor(((intersection[intIndex].agingTime[0] *
individual_phase_queue_lengths[0]) + (intersection[intIndex].agingTime[4] *
individual_phase_queue_lengths[4]))/(MAX(individual_phase_queue_lengths[0] +
individual_phase_queue_lengths[4], 1))); //NEMA dual phase 1/5
	intersection[intIndex].agingFactor[1] =
CalculateAgingFactor(((intersection[intIndex].agingTime[0] *
individual_phase_queue_lengths[0]) + (intersection[intIndex].agingTime[5] *
individual_phase_queue_lengths[5]))/(MAX(individual_phase_queue_lengths[0] +
individual_phase_queue_lengths[5], 1))); //NEMA dual phase 1/6
	intersection[intIndex].agingFactor[2] =
CalculateAgingFactor(((intersection[intIndex].agingTime[1] *
individual_phase_queue_lengths[1]) + (intersection[intIndex].agingTime[4] *
individual_phase_queue_lengths[4]))/(MAX(individual_phase_queue_lengths[1] +
individual_phase_queue_lengths[4], 1))); //NEMA dual phase 2/5
	intersection[intIndex].agingFactor[3] =
CalculateAgingFactor(((intersection[intIndex].agingTime[1] *
individual_phase_queue_lengths[1]) + (intersection[intIndex].agingTime[5] *
individual_phase_queue_lengths[5]))/(MAX(individual_phase_queue_lengths[1] +
individual_phase_queue_lengths[5], 1))); //NEMA dual phase 2/6
	intersection[intIndex].agingFactor[4] =
CalculateAgingFactor(((intersection[intIndex].agingTime[2] *
individual_phase_queue_lengths[2]) + (intersection[intIndex].agingTime[6] *
individual_phase_queue_lengths[6]))/(MAX(individual_phase_queue_lengths[2] +
individual_phase_queue_lengths[6], 1))); //NEMA dual phase 3/7
	intersection[intIndex].agingFactor[5] =
CalculateAgingFactor(((intersection[intIndex].agingTime[2] *
individual_phase_queue_lengths[2]) + (intersection[intIndex].agingTime[7] *
individual_phase_queue_lengths[7]))/(MAX(individual_phase_queue_lengths[2] +
individual_phase_queue_lengths[7], 1))); //NEMA dual phase 3/8
	intersection[intIndex].agingFactor[6] =
CalculateAgingFactor(((intersection[intIndex].agingTime[3] *
individual_phase_queue_lengths[3]) + (intersection[intIndex].agingTime[6] *
individual_phase_queue_lengths[6]))/(MAX(individual_phase_queue_lengths[3] +
individual_phase_queue_lengths[6], 1))); //NEMA dual phase 4/7
```

```
	intersection[intIndex].agingFactor[7] =
CalculateAgingFactor(((intersection[intIndex].agingTime[3] *
individual_phase_queue_lengths[3]) + (intersection[intIndex].agingTime[7] *
individual_phase_queue_lengths[7]))/(MAX(individual_phase_queue_lengths[3] +
individual_phase_queue_lengths[7], 1))); //NEMA dual phase 4/8
}

void qpx_CLK_startOfSimLoop()
{
	float currentTime = qpg_CFG_simulationTime(); //elapsed simulation time in
seconds
	int i;

	if ((currentTime > 3*DELTA) && (currentTime < 5*DELTA)) //set to correct
lights after the first time step
	{
		qps_GUI_printf("initial phase entered at time %f \n", currentTime);
		//set initial priorities, 1/5 green

		for (i = 0; i < numberOfIntersections; i++)
		{
			qps_LNK_priority(intersection[i].NORTH_in,
intersection[i].WEST_out, APIPRI_MINOR);  //4
			qps_LNK_priority(intersection[i].NORTH_in,
intersection[i].SOUTH_out, APIPRI_BARRED); //4
			qps_LNK_priority(intersection[i].NORTH_in,
intersection[i].EAST_out, APIPRI_BARRED);  //7

			qps_LNK_priority(intersection[i].EAST_in,
intersection[i].NORTH_out, APIPRI_BARRED);  //6
			qps_LNK_priority(intersection[i].EAST_in,
intersection[i].WEST_out, APIPRI_BARRED);   //6
			qps_LNK_priority(intersection[i].EAST_in,
intersection[i].SOUTH_out, APIPRI_MAJOR);  //1

			qps_LNK_priority(intersection[i].SOUTH_in,
intersection[i].EAST_out, APIPRI_MINOR);  //8
			qps_LNK_priority(intersection[i].SOUTH_in,
intersection[i].NORTH_out, APIPRI_BARRED); //8
			qps_LNK_priority(intersection[i].SOUTH_in,
intersection[i].WEST_out, APIPRI_BARRED);  //3

			qps_LNK_priority(intersection[i].WEST_in,
intersection[i].SOUTH_out, APIPRI_BARRED);  //2
			qps_LNK_priority(intersection[i].WEST_in,
intersection[i].EAST_out, APIPRI_BARRED);  //2
			qps_LNK_priority(intersection[i].WEST_in,
intersection[i].NORTH_out, APIPRI_MAJOR); //5
		}
	}

	//uncomment this following section for the aging factors to be calculated
(NOTE: this is the corridor  version)
```

```
	if (fabs(currentTime - (int)currentTime) < DELTA) //update aging time and
aging factor, prior to calling the IMA
	{
		for (i = 0; i < numberOfIntersections; i++)
		{
			//sequence of these functions is important 1)TimeLastServed,
2) AgingTime, 3) AgingFactor
			UpdateTimeLastServed(currentTime, i);
			UpdateAgingTime(currentTime, i);
			//UpdateAgingFactors(i);
			UpdateAgingFactors2(i);
		}
	}

		IMA2("5"); //start calling immediately, initial phase is 1/5, check
if vehicles are still spawning, //Call IMA2 for each intersection in the network
		IMA2("m5");
		IMA2("n5");

	if (fabs(currentTime - (int)currentTime) < DELTA) //update queuesDischarged
once a second, //update current delay once a second
	{
		//check queueTailIDs array to update queuesDischarged boolean
		if (currentTime > (minimumGreenDuration + 4)) //prevent check from
occurring during the first phase of the simulation (as there are no queues to
check at the beginning of the simulation), NOTE: need to add logic to accomodate
larger networks with additional intersections
		{
			for (i = 0; i < numberOfIntersections; i++)
			{
				intersection[i].queuesDischarged =
QueuesDischargedCheck(i, ALLPHASES[intersection[i].currentPhaseIndex]); //what
about for the first phase of the simulation?
			}
		}
	}

	//update lastPhaseIndex
	for (i = 0; i < numberOfIntersections; i++)
	{
		intersection[i].lastPhaseIndex = intersection[i].currentPhaseIndex;
//since this is at the beginning of the time step, currentPhaseIndex acutally
refers to the resulting calculations of the previos time step
	}
	//qps_GUI_printf("qpx_CLK_startOfSimLoop() entered at time %f , DELTA = %f
\n", currentTime, DELTA);
}

void qpx_CLK_endOfSimLoop()
{
	int i = 0;
	float currentTime = qpg_CFG_simulationTime(); //elapsed simulation time in
seconds
```

```
	if (fabs(currentTime - (int)currentTime) < DELTA) //update aging time and
aging factor
	{
		for (i = 0; i < numberOfIntersections; i++)
		{
			// update current delay for all 8 dual phases
			intersection[i].PhaseCurrentDelay[0] =
Find_Phase_Current_Delay0(i, ALLPHASES[0]);
			intersection[i].PhaseCurrentDelay[1] =
Find_Phase_Current_Delay0(i, ALLPHASES[6]);
			intersection[i].PhaseCurrentDelay[2] =
Find_Phase_Current_Delay0(i, ALLPHASES[12]);
			intersection[i].PhaseCurrentDelay[3] =
Find_Phase_Current_Delay0(i, ALLPHASES[18]);
			intersection[i].PhaseCurrentDelay[4] =
Find_Phase_Current_Delay0(i, ALLPHASES[24]);
			intersection[i].PhaseCurrentDelay[5] =
Find_Phase_Current_Delay0(i, ALLPHASES[30]);
			intersection[i].PhaseCurrentDelay[6] =
Find_Phase_Current_Delay0(i, ALLPHASES[36]);
			intersection[i].PhaseCurrentDelay[7] =
Find_Phase_Current_Delay0(i, ALLPHASES[42]);
		}
	} //apparently qpx_DRW_ModelView is called after this function at the end
of the time step
}

float in_qpo_CFM_Speed(LINK* link, VEHICLE* Vp, float CFM_Speed)
{
	float v_lmt = qpg_LNK_speedlimit(link)/2.2369 + 10; //NOTE:
qpg_LNK_speedlimit(link) returns speed limit in mph
	float speedLimit = qpg_LNK_speedlimit(link)/2.2369; //	divisor
converts from mph to m/s
	float currTime = qpg_CFG_simulationTime();
	float calSpeed, recSpeed, currVel = qpg_VHC_speed(Vp); //calSpeed is used
for ESH, NOTE: qpg_VHC_speed(Vp) returns vehicle speed in m/s
	int linkid = qpg_LNK_index(link), mode, currZone =
qpg_ZNE_externalIndex(qpg_LNK_zone(link));
	float timeStep = qpg_CFG_timeStep();

	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp); // members of
current vehicle (pointer is from input parameter)
	float acceleration = 0; // used for acceleration color scheme
	float colorScale = 0;

	// calculate emissions
	if (fabs(currTime - (int)(currTime+0.5)) > DELTA || currZone == myVeh-
>origin || currZone == myVeh->dest) //these are conditions under which emissions
should not be calculated
		goto CACC;
	if (myVeh->first2sec < 3)
	{
		myVeh->tm[myVeh->first2sec] = currTime;
		myVeh->linkID[myVeh->first2sec] = linkid;
```

```
		myVeh->vel[myVeh->first2sec] = currVel;
		myVeh->first2sec++;
		goto CACC;
	}
	calEmissions(myVeh, currTime, currVel, linkid); //emissions are calculated
every three seconds

CACC:
	//Update current network delay for vehicle
	//NOTE: all return values from paramics for speed are in meters/second
	myVeh->currentNetworkDelay += (timeStep * (1 - (MIN(currVel,
speedLimit)/speedLimit))); //max function is to account for case of vehicle speed
being above the speed limit



	return CFM_Speed;

}

float qpo_CFM_leadSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
	float CFM_Speed = qpg_CFM_leadSpeed(link, Vp, ahead);
	return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
}

float qpo_CFM_followSpeed(LINK* link, VEHICLE* Vp,  VEHICLE* ahead[])
{
	float CFM_Speed = qpg_CFM_followSpeed(link, Vp, ahead);
	return in_qpo_CFM_Speed(link, Vp, CFM_Speed);
}

void qpx_VHC_arrive(VEHICLE* Vp, LINK* link, ZONE* zone)
{
	VEHICLE_DATA *myVeh = (VEHICLE_DATA*)qpg_VHC_userdata(Vp);
	TT_veh[counter_arrived] = qpg_VHC_existTime(Vp);
	VMT += myVeh->total_dist; //myVeh->tripDist; //tripDist was originally used
here (when RTR was working) ...try total_dist, check if it is the same as tripDist
with RTR (if RTR is not bugged)
	VHT += TT_veh[counter_arrived];

	if ((myVeh->origin == 1 && myVeh->dest == 2) || (myVeh->origin == 2 &&
myVeh->dest == 1)) //must be a mainstream corridor vehicle
	{
		VMT_c += myVeh->total_dist;
		VHT_c += TT_veh[counter_arrived];
		counter_arrived_c++;
	}
	else //must be a vehicle not on the coordinated phase
	{
		VMT_nc += myVeh->total_dist;
		VHT_nc += TT_veh[counter_arrived];
		counter_arrived_nc++;
	}
```

```
	counter_arrived++;
	free(myVeh);
}

void  qpx_NET_complete()
{
	FILE *fsbs, *fsum, *fopMode, *fVSP, *fTT, *fsbs_e, *fsum_e, *fopMode_e,
*fVSP_e, *fTT_e, *fsbs_n, *fsum_n, *fopMode_n, *fVSP_n, *fTT_n;
	int i;

	VMT = VMT*meter2mile;
	dist_sum = dist_sum*meter2mile;
	energy_sum = energy_sum/dist_sum/3600;
	CO2_sum = CO2_sum/dist_sum/3600;
	CO_sum = CO_sum/dist_sum/3600;
	HC_sum = HC_sum/dist_sum/3600;
	NOx_sum = NOx_sum/dist_sum/3600;
	PM_sum = PM_sum/dist_sum/3600;

	if (counter_arrived_c > 0)
	{
		VMT_c = VMT_c*meter2mile;
		dist_sum_c = dist_sum_c*meter2mile;
		energy_sum_c = energy_sum_c/dist_sum_c/3600;
		CO2_sum_c = CO2_sum_c/dist_sum_c/3600;
		CO_sum_c = CO_sum_c/dist_sum_c/3600;
		HC_sum_c = HC_sum_c/dist_sum_c/3600;
		NOx_sum_c = NOx_sum_c/dist_sum_c/3600;
		PM_sum_c = PM_sum_c/dist_sum_c/3600;
	}
	if (counter_arrived_nc > 0)
	{
		VMT_nc = VMT_nc*meter2mile;
		dist_sum_nc = dist_sum_nc*meter2mile;
		energy_sum_nc = energy_sum_nc/dist_sum_nc/3600;
		CO2_sum_nc = CO2_sum_nc/dist_sum_nc/3600;
		CO_sum_nc = CO_sum_nc/dist_sum_nc/3600;
		HC_sum_nc = HC_sum_nc/dist_sum_nc/3600;
		NOx_sum_nc = NOx_sum_nc/dist_sum_nc/3600;
		PM_sum_nc = PM_sum_nc/dist_sum_nc/3600;
	}

	fsum = fopen(out2Path, "w");
	fsum_e = fopen(out21Path, "w");
	fsum_n = fopen(out22Path, "w");

	if (fsum == NULL || fsum_e == NULL || fsum_n == NULL) //never actually
executes
	{
		qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);
```

```
		qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);

		qps_GUI_printf("Couldn't open file. (simulation ended, results not
logged)");
	}

	fprintf(fsum,
"Energy(KJ/mi),CO2(g/mi),CO(g/mi),HC(g/mi),NOx(g/mi),PM2.5(g/mi),VHT(sec/veh),VMT(
mi/veh)\n");
	if (counter_arrived > 0)
	{
		fprintf(fsum, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum,
CO2_sum, CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived,
VMT/counter_arrived);
		fprintf(fsum, "%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
	}

	if (counter_arrived_c > 0)
	{
		fprintf(fsum, "Coordinated Phase Vehicle Statistics: \n");
		fprintf(fsum, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_c,
CO2_sum_c, CO_sum_c, HC_sum_c, NOx_sum_c, PM_sum_c, VHT_c/counter_arrived_c,
VMT_c/counter_arrived_c);
		fprintf(fsum, "%d coordinated phase vehicles released. %d
coordinated phase vehicles arrived.\n\n", counter_released_c,counter_arrived_c);
	}

	if (counter_arrived_nc > 0)
	{
		fprintf(fsum, "Non-Coordinated Phase Vehicle Statistics: \n");
		fprintf(fsum, "%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum_nc,
CO2_sum_nc, CO_sum_nc, HC_sum_nc, NOx_sum_nc, PM_sum_nc,
VHT_nc/counter_arrived_nc, VMT_nc/counter_arrived_nc);
		fprintf(fsum, "%d non-coordinated phase vehicles released. %d non-
coordinated phase vehicles arrived.\n", counter_released_nc,counter_arrived_nc);
	}

	fclose(fsum);
	fclose(fsum_e);
	fclose(fsum_n);

	qps_GUI_printf("%f, %f, %f, %f, %f, %f, %f, %f\n", energy_sum, CO2_sum,
CO_sum, HC_sum, NOx_sum, PM_sum, VHT/counter_arrived, VMT/counter_arrived);
	qps_GUI_printf("	coordinated phase vehicle statistics: %f, %f, %f,
%f, %f, %f, %f, %f\n", energy_sum_c, CO2_sum_c, CO_sum_c, HC_sum_c, NOx_sum_c,
PM_sum_c, VHT_c/counter_arrived_c, VMT_c/counter_arrived_c);
	qps_GUI_printf("non-coordinated phase vehicle statistics: %f, %f, %f, %f,
%f, %f, %f, %f\n", energy_sum_nc, CO2_sum_nc, CO_sum_nc, HC_sum_nc, NOx_sum_nc,
PM_sum_nc, VHT_nc/counter_arrived_nc, VMT_nc/counter_arrived_nc);
```

```
        qps_GUI_printf("%d vehicles released. %d vehicles arrived.\n",
counter_released,counter_arrived);
}
```