UNIVERSITY OF CALIFORNIA, SAN DIEGO

Enhancing networking protocols in widely deployed devices

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Patrick Verkaik

2010

The dissertation of Patrick Verkaik is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2010

DEDICATION

To my grandparents, for many fond memories.

EPIGRAPH

It is not the strongest of the species that survives,
nor the most intelligent that survives.
It is the one that is the most adaptable to change.
—Charles Darwin

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I have more people to thank than I can possibly fit into a few pages, and unfortunately, likely more than I can remember. So please bear with me.

First and foremost, I'd like to thank my advisor, Alex Snoeren, for being my main mentor and teacher during my time at the Computer Science department at UCSD, for giving me so much freedom to pursue topics of my choosing, and for enduring my late schedule. As one of the espresso czars, I'd also like to thank him, on behalf of all coffee drinkers in our department, for providing us with an espresso machine that has kept us caffeinated over the years, and will hopefully do so for many years more! I'd like to thank Alex, as well as Geoff Voelker, kc claffy, Rajesh Gupta, and Tara Javidi for being on my doctoral committee, and for their patience in receiving thesis chapters, allowing me to take time planning the next steps of my career.

I am indebted to a number of people who were essential in my career path leading up to the Ph.D. program: Maarten van Steen, kc claffy, Frances Brazier, Teus Hagen, Daniel Karrenberg, Ted Lindgreen, Evi Nemeth, Roland van der Pol, and Wytze van der Raay — it is only thanks to the personal efforts of these people that I ultimately chose, and was able, to enter the Ph.D. program. My time at CAIDA was very special to me, with people like Nevil Brownlee, kc claffy, Brad Huffaker, Young Hyun, David Moore, and Colleen Shannon serving as a great source of inspiration to me.

Before and during my Ph.D. program it has been my pleasure to collaborate with many, many people: Mikhail Afanasyev, Sharad Agarwal, Yuvraj Agarwal, Egon Amade, Victor Bahl, Arno Bakker, Gerco Ballintijn, Peter Benko, Frances Brazier, Andre Broido, Yu-Chung Cheng, Jennifer Chiang, kc claffy, Ruomei Gao, Rajesh Gupta, Young Hyun, Marten Jansen, Srikanth Kandula, Eelco Klaver, Ihor Kuz, Ratul Mahajan, Kobus van der Merwe, Benno Overeinder, Jitu Padhye, Dan Pei, Roland van der Pol, Barath Raghavan, Stefan Savage, Tom Scholl, Aman Shaikh, Henk Sips, Alex Snoeren, Oliver Spatscheck, Maarten van Steen, Andy Tanenbaum, Geoff Voelker, and Ivo van der Wijk. Much of what I have learned in these last years is from my collaborators, and I am grateful for that.

A CS researcher can do little without a (sys)admin's help. Thanks to Dan Andersen, Jocelyn Bernardo, Julie Conner, Grant DuVall, Chris Edwards, Brian Kantor, Alex Ma, Marvin McNett, Michelle Panik, Darlene Piche, Kathryn Reed, Betty Tso, Dave Wargo, Bill Young and everyone at CSE Help for making this student's life so much easier! Also, Mikhail Afanasyev and Brian Kantor's knowledge of electrical engineering has proved indispensable on many occasions.

I have benefited from feedback on papers and talks from Mikhail Afanasyev, Yuvraj Agarwal, Alvin AuYoung, Diwaker Gupta, Keon Jang, Priya Mahadevan, Justin Ma, Harsha Madhyastha, John McCullough, Sue Moon, Radhika Mysore, Barath Raghavan, Stefan Savage, Amin Vahdat, Kashi Vishwanath, Geoff Voelker, Michael Vrable, and Qing Zhang. Many thanks for that! I would further like to acknowledge the help of Sharad Agarwal, Mukesh Agrawal, Yu-Chung Cheng, kc claffy, John Dunagan, Alex Loukissas, Ratul Mahajan, Shaan Mahbubani, Kobus van der Merwe, Stefan Savage, Alex Snoeren, George Varghese, and Geoff Voelker, for their advice, reference letters, and pointers, while I was planning the next steps of my career.

I couldn't have wished for a better office environment at UCSD than the one I was part of. The mix of personalities of my labmates over the years — Alvin AuYoung, Ryan Braud, Justin Ma, Damon McCoy, Andreas Pitsillidis, Barath Raghavan, and Michael Vrable — was a great source of discussions and humor, offensive as well as inoffensive. While not an office member, Mikhail's presence was always felt through his nerf gun shooter and the many other gadgets he constructed and scattered throughout our office. There are many others in the Ph.D. program I would like to thank for making my time at UCSD so enjoyable, but the list would be too long. However, my acknowledgements would be incomplete without mentioning Jeff Brown, Jocelyne Bruand, Yu-Chung Cheng, Emi Curtmola, Banu Dost, Shaan Mahbubani, John McCullough, Marti Motoyama, Radhika Mysore, Malveeka Tewari, and Bhanu Vattikonda.

I would never have survived without the companionship and advice of my friends, far and near. I'd like to thank my friends from the Netherlands (Peter, Tycho, Wouter), my friends from I-House (Alessia, Chris, Gina, Grace, Luca, Naoko, Neuman, On, Rishi, Toni, Viviana), the Takens (Didem, Gjergji, Marisol, Mary, Nikos, Panos, Tikir, Winnie, Yuvraj). Fontas, Jens, and Johnnie are each in their own category.

Finally, I'd like to thank the people who are the most dear to me: my parents and my sister for letting me travel halfway across the world to pursue my dreams; Ruomei for her insights, her patience, and, most of all, her unwavering affection.

Post-finally, I should acknowledge the following reprints of published material used in my dissertation. Chapter 2, in part, is a reprint of the material as it appears in Proceedings of the 6th ACM/USENIX Symposium on Networked Systems Design and

Implementation (NSDI), Boston, MA, April 2009. Verkaik, Patrick; Agarwal, Yuvraj; Gupta, Rajesh; Snoeren, Alex C. Chapter 4, in part, is a reprint of the material as it appears in Proceedings of the USENIX Annual Technical Conference, pages 295–308, Santa Clara, CA, June 2007. Verkaik, Patrick; Pei, Dan; Scholl, Tom; Shaikh, Aman; Snoeren, Alex C.; Van der Merwe, Jacobus. Appendix A, in part, is a reprint of the material as it appears in Proceedings of the USENIX Annual Technical Conference, pages 295–308, Santa Clara, CA, June 2007. Verkaik, Patrick; Pei, Dan; Scholl, Tom; Shaikh, Aman; Snoeren, Alex C.; Van der Merwe, Jacobus. The dissertation author was the primary investigator and author of these papers.

VITA

1996	M. S. in Computer Science *cum laude*, Vrije Universiteit, Amsterdam
2010	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

“Secure and Policy-Compliant Source Routing.” Barath Raghavan, Patrick Verkaik, Alex C. Snoeren. *IEEE/ACM ToN.* August 2009.

“Detailed Diagnosis in Enterprise Networks.” Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitu Padhye, Paramvir Bahl. *ACM SIGCOMM.* August 2009.

“Softspeak: Making VoIP Play Well in Existing 802.11 Deployments.” Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta, Alex C. Snoeren. *ACM/USENIX NSDI.* April 2009.

“Web Service Access Management for Integration with Agent Systems.” Benno J. Overeinder, Patrick Verkaik, Frances M. T. Brazier. *ACM Symposium On Applied Computing.* March 2008.

“Automating Cross-Layer Diagnosis of Enterprise Wireless Networks.” Yu Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Peter Benko, Jennifer Chiang, Alex C. Snoeren, Geoffrey M. Voelker, Stefan Savage. *ACM SIGCOMM.* August 2007.

“Wresting Control from BGP: Scalable Fine-grained Route Control.” Patrick Verkaik, Dan Pei, Tom Scholl, Aman Shaikh, Alex C. Snoeren, Jacobus van der Merwe. *USENIX Annual Technical Conference.* June 2007.

“PRIMED: A Community-of-Interest-Based DDoS Mitigation System.” Patrick Verkaik, Oliver Spatscheck, Jacobus van der Merwe, Alex C. Snoeren. *ACM SIGCOMM LSAD Workshop.* September 2006.

“Global Distribution of Free Software (and other things).” Arno Bakker, Ihor Kuz, Maarten van Steen, Andrew S. Tanenbaum, Patrick Verkaik. *System Administration and Network Engineering.* May 2002.

“Encapsulating Distribution in Remote Objects.” Marten Jansen, Eelco Klaver, Patrick Verkaik, Maarten van Steen, Andrew S. Tanenbaum. *Information and Software Technology.* May 2001.

“A Distributed-Object Infrastructure for Corporate Websites.” Ihor Kuz, Patrick Verkaik, Maarten van Steen, Henk J. Sips. *Distributed Objects and Applications.* September 2000.

The Globe Distribution Network.” Arno Bakker, Egon Amade, Gerco Ballintijn, Ihor Kuz, Patrick Verkaik, Ivo van der Wijk, Maarten van Steen, Andrew S. Tanenbaum. *USENIX Annual Technical Conference (FREENIX Track).* June 2000.

ABSTRACT OF THE DISSERTATION

Enhancing networking protocols in widely deployed devices

by

Patrick Verkaik

Doctor of Philosophy in Computer Science

University of California, San Diego, 2010

Professor Alex C. Snoeren, Chair

Standardization of networking protocols enables interoperability and reduces cost, but is generally a slow process. As a result, there are many cases in which a networking protocol and the devices that implement it are not able to satisfy emerging user requirements. In such cases, the networking protocol falls short of current user requirements but has been too widely deployed to make a redesign sufficiently timely or cost-effective. In this dissertation, we advocate an approach to solving this problem that leaves deployed devices unmodified, by treating them as a black box. Rather, we show that for certain network devices, it may be sufficient to change the behavior of the device by "externally controlling" it, *e.g.*, from software or another device, such that it solves the problem at hand.

We demonstrate the effectiveness of such a black box approach for several challenging problems from two different areas: two problems from 802.11 wireless networking and one problem from inter-domain routing. In the first problem, we discuss the increasingly popular use of VoIP applications in the 802.11 protocol. Our experimental results show that 802.11 is ill-designed to carry VoIP traffic: an 802.11 hotspot faced with even a moderate number of VoIP calls in combination with data traffic may severely degrade the performance of both. The second problem, also from the area of 802.11, addresses unfairness of 802.11 in the face of emerging and potentially selfish upload-oriented applications. The third problem is from the area of inter-domain routing. We address a modern ISP's requirement for detailed control over routing to meet traffic engineering and security goals and to support applications such as VoIP. We show that the current inter-domain routing protocol, BGP, lacks the flexibility needed to implement the required degree of route control.

We have implemented solutions to these problems that follow a black box approach and evaluated their effectiveness in combination with commercial networking devices. Our evaluation demonstrates that enhancing deployed networking protocols using a black box approach can be effective and efficient, yet portable and backward compatible with legacy devices.

Chapter 1

Introduction

A wide variety of networking technology exists today that implements standardized protocols, enabling interoperability and commoditization of networking devices, competition among vendors, and generally reducing cost. However, standardization is a slow process, and as a result we may expect that in many cases a standardized networking protocol and the devices that implement the protocol are not able to satisfy emerging user requirements. As an example, recent years have seen the emergence of a large number of cell phone models equipped with both 802.11 (WiFi) and VoIP (Voice over IP) capabilities, giving users the ability to place inexpensive calls when in range of a WiFi hotspot. Consequently, we expect 802.11 networks to increasingly be required to serve voice users in addition to data (*e.g.*, Web) users. Unfortunately, it is known that 802.11 was not designed to efficiently carry a mix of voice and data traffic, and therefore data and voice users sharing a Starbucks WiFi hotspot may find that neither type of user receives satisfactory performance. Surprisingly, adequate solutions to this problem have been known for some time, yet are not available to users today. In such cases, users are left to choose between waiting for a future cycle of standardization and subsequent re-implementation and redeployment of the devices to catch up with current requirements (a costly and lengthy process), or alternatively buying an expensive custom-made device, eliminating many of the advantages of standardization.

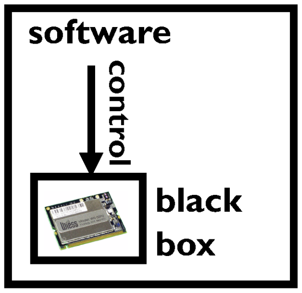
The above is an instance of a more general phenomenon: a protocol (802.11), as standardized and implemented today, may not be able to match an emerging user need (to efficiently carry a mix of VoIP and data traffic). An important part of the problem is the complex nature of the standardization process, which requires consensus among committee members with diverse interests, leading to long delays. For example, the average time for an initial Internet Draft to become an IETF RFC is 2.5 years [64]. In the case of IEEE, recent standardization efforts of 802.11 (e and n) have each taken well over four years to complete [3, 4]. While a committee based process might appear to impose unacceptable delays on standardization, a game theoretic comparison of standardization processes [28] suggests that a committee based process is likely to lead to higher quality standards than an uncoordinated, market-based process, even when factoring in the cost of delay. Thus, it is reasonable to expect that standardization will remain a lengthy undertaking, and therefore standards will continue to lag behind user demand by a significant amount of time.

In this dissertation we address the challenge of letting a *current* generation of a standardized protocol, as well as the networking devices that implement it, satisfy *current* user needs. The ability to do so allows users to evolve their needs and deployments independent of the slow pace of standardization. Unless protocol and hardware designers develop techniques that allow protocols and hardware to be extensible enough to anticipate future user needs, this ability is likely to remain important.

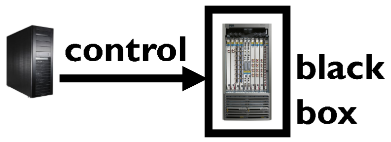
1.1 Approach

While there are many ways to address this challenge, in this dissertation, we advocate a 'black box approach.' Continuing the above example, consider Samsung, one of many manufacturers of smartphones with 802.11 and VoIP capabilities. A WiFi (802.11) phone contains software as well as various hardware components, among them an 802.11 network card likely manufactured by a third party.[1] If Samsung wants to fix 802.11 to better support VoIP for its customers, it needs to convince an IEEE standards committee to change 802.11 and/or its third party supplier to redesign the 802.11 card. Furthermore, if Samsung wants to make improvements for its existing customers, it needs to recall millions of phones that have already been sold in order to update them. Therefore, redesigning 802.11 and/or re-implementing the 802.11 card is very likely a time-consuming and expensive proposition. We therefore restrict the solution space to those solutions that incorporate the existing 802.11 hardware (and therefore the protocol) unmodified. Specifically, in our black box approach, rather than changing the *implementation* of the 802.11 card, Samsung modifies the card's *behavior* by carefully

[1]In reality, this may be an 802.11 chip rather than a card.



(a)



(b)

Figure 1.1: Instances of externally controlling a networking device. (a) Controlling a device (a network card) from software through a hardware interface. (b) Controlling a device (a router) from a server through a networking protocol.

'externally controlling' its inputs from software (Figure 1.1(a)). Note that from the point of view of Samsung, the device to be controlled is the 802.11 card, rather than the phone. Assuming that such an approach can indeed be effective, all Samsung now needs to do to deploy its solution is to modify the phone software and issue a software update.

Can such a black box solution indeed be effective? We devise three criteria that we claim are necessary for that to be the case, illustrated in Figure 1.2. Since the approach treats the device as a black box, these criteria constrain the various ways the controlled device interacts with the outside world. The first two criteria (*sufficient control* and *portability*) constrain the choice of interface used to control the device. The third criterion (*protocol-feasibility*) constrains the interaction of the controlled device with other devices. Our first criterion is that the interface to the device (in this case, the 802.11 card) provides *sufficient control* to implement the solution, both in terms of functionality and of performance. Several such interfaces may exist, and, in our example, Samsung needs to decide which interface to use to effect the changes. Our second criterion constrains this choice by requiring the interface to be *portable* across devices from multiple vendors. Let's assume Samsung uses different 802.11 card suppliers for its various WiFi phone models (or wants to keep open the option of switching supplier in the future). Clearly, if Samsung chooses a portable interface, one supported by any 802.11 card, it may reduce the complexity of its phone software and thereby its development cost. An interface can be portable due to having been standardized (*e.g.*, USB) or because it exploits features that are 'typically' available in the class of device (*e.g.*, many 802.11 cards allow their channel to be fixed). Finally, while Samsung wants to improve

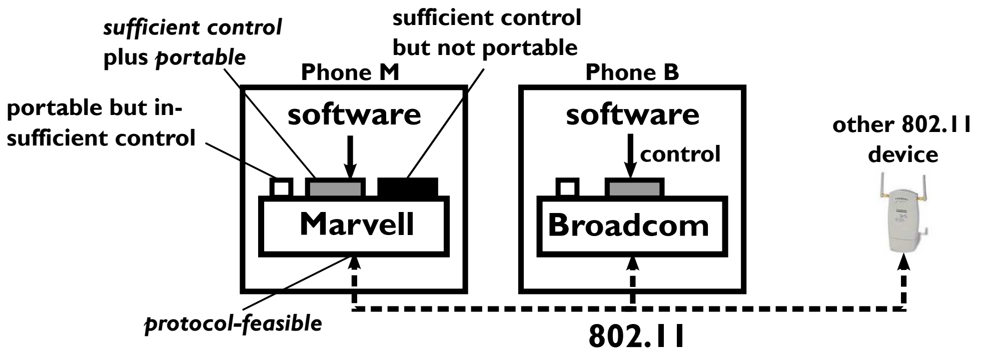

sufficient control plus portable
sufficient control but not portable
Phone M
Phone B
portable but in-sufficient control
software
software
control
other 802.11 device
Marvell
Broadcom
protocol-feasible
802.11

Figure 1.2: Three criteria for an effective black box approach. Two phone models, M and B, are show, both made by the same manufacturer but each based on different 802.11 hardware, Marvell and Broadcom. The white and grey interfaces are supported by both types of 802.11 hardware and are thus *portable*. However, of these only the grey interface gives *sufficient control*, and thus the software in both phone models uses that to externally control the 802.11 hardware. The 802.11 card is controlled in such a way that it remains *protocol-feasible* and can interact with other 802.11 devices.

the performance of its WiFi phone, its customers will be unhappy if their phones stop interoperating with other 802.11 hardware (access points, laptops, other phones, ...). Since Samsung is not changing the 802.11 protocol, any changes it makes in the 802.11 card behavior need to be backward compatible with existing 802.11 hardware. Therefore, our third criterion is that the modified behavior of the device is *protocol-feasible*, *i.e.*, permitted by the protocols implemented by the device. For example, having an 802.11 card send a packet without preamble is not feasible, since other 802.11 devices need the preamble to prepare for packet reception. On the other hand, letting the card back off by a fixed rather than a random amount of time is feasible, since a compliant 802.11 receiver does not distinguish between the two.

To understand whether a given solution uses an effective black box approach, we can evaluate it using these three criteria. We evaluate *protocol-feasibility* by inspecting the protocol specification and experimentally testing interoperability with other devices. Evaluating *sufficient control* typically consists of a performance evaluation that asks whether an implementation of the solution efficiently achieves the design goals. We consider *portability* to be a more subjective criterion: essentially it is a measure of the expected amount of work needed to implement the solution across different devices.

Given that a black box approach imposes a constraint on the design of a solution, we may expect it to have some cost compared to an unconstrained, clean slate approach. In particular, a designer following a clean slate approach may choose to modify the device and/or the protocol, rather than implement a black box solution, for several reasons. First, a clean slate solution may perform more efficiently, a cost that is captured by the sufficient control criterion. Second, a black box solution may result in a more complex solution than a clean slate solution, imposing additional development cost and increasing the potential for bugs. Third, a successful black box solution may hinder the deployment of a more effective clean slate solution, similar to the manner in which the success of Network Address Translation may have partly been responsible for delaying the deployment of IPv6 [82]. These costs must be weighed against the costs of a clean slate approach (as discussed earlier). In cases where a clean slate approach is ultimately preferable, a hybrid approach may combine the best of both worlds: while a particular black box solution might be a temporary solution that will eventually be replaced by the clean slate solution, it may yet provide valuable deployment experience guiding the design of the clean slate solution.

1.2 Contributions

We claim that a black box approach can be used to enhance deployed networking protocols to solve a diverse set of problems. This dissertation demonstrates this point using three concrete problems that vary across several dimensions, each of which presents a specific challenge for a black box approach. First, in Chapter 2 we consider the challenge of externally controlling 802.11 cards to efficiently support VoIP traffic, requiring *fine-grained timing*. In 802.11, events occur at a small enough time scale that the protocol is typically implemented in hardware. Nevertheless, we show that it is possible to effectively control 802.11 hardware from software to create a fine-grained TDMA (time-division multiple access) scheme that significantly improves the performance of VoIP. Next, in Chapter 3 we tackle unfairness of 802.11 in the face of emerging and potentially selfish upload-oriented applications. We devise a solution in which an access point externally controls 802.11 clients that is robust against certain types of *adversarial behavior*. Third, in Chapter 4, we show how a large ISP may externally control its router base, allowing the ISP to implement powerful customer-oriented services. Here, given the size of an ISP's router base, the challenge is to perform external control at *large scale*. In addition, whereas in Chapter 2 we implement our black box approach by externally controlling a device from software, the solutions in Chapters 3 and 4 require devices (*e.g.*, an 802.11 client or a router) to be controlled from a separate device (an access point or a server, resp.). Figure 1.1(b) illustrates the case of a server controlling a router.

In all cases, the decentralized nature of the underlying protocol (802.11 or the Border Gateway Protocol) presents an additional challenge for a black box approach, since by definition the protocol does not identify a single point of control. Note that this is true even for access point based deployments of 802.11, since access points do not control the manner in which stations access the channel.

1.3 Related work

Previous work has advocated the use of 'gray box techniques' and a 'gray toolbox' [11], similar in spirit to our black box approach. The purpose of the gray toolbox is to augment an operating system (OS) with OS like services implemented in user space, treating the OS as a black box that need not be modified. As an example, the authors design a service that infers the contents of the OS's file cache, which applications can then use to reorder their data accesses for better performance. While at a high level their approach is similar to ours, the goal of our work is to enhance *network* protocols and their implementations, which raises a different set of challenges and opportunities. For example, when applying a black box approach to network devices, one must consider compatibility of the device with other network devices (protocol feasibility). In addition, a network based black box approach allows a designer to take advantage of cross-device control (Figure 1.1(b)).

In recent work, OpenFlow [48] defines an interface that allows an OpenFlow-enabled switch to be externally controlled. For example, using OpenFlow, Portland [51] controls forwarding in switches in order to redesign data center networking using a black box approach. OpenFlow is an exciting development that without doubt will substantially ease future development of black box solutions for switching technology. However, a fundamental limitation of an OpenFlow like approach is that it can only meet user demands similar to those that have already been identified. A black box approach remains essential for handling other user requirements (subject to the criteria for an effective black box approach above). Similar to the way a black box solution can be used to guide the design of clean slate approaches, we believe that it can also guide the design of future

OpenFlow like interfaces.

Finally, many instances of a black box approach have appeared in the literature, but to our knowledge their authors do not articulate general design principles for a black box approach as we do. We discuss several of them [16, 67, 75] in the chapters ahead, evaluating them based on our black box criteria. By contrast, we devise black box solutions to problems characterized by a diverse set of challenges — fine-grained timing in Chapter 2, adversarial behavior in Chapter 3, and large scale in Chapter 4 — explicitly using our criteria to both guide the design of our solutions and evaluate their effectiveness as black box solutions.

Chapter 2

Softspeak

Having outlined the general approach proposed by the thesis, we now turn to our first case study: the performance of Voice-over-IP (VoIP) in 802.11/WiFi[1]. VoIP technology is now pervasive in wire-line networks, embodied by wildly successful applications like Skype. Wireless deployment, in contrast, has so far been limited to certain niche products. Recently, however, WiFi-capable consumer phone handsets such as T-Mobile's UMA and the Apple iPhone have been released to the US market in large numbers, portending a huge influx of WiFi VoIP users as third-party applications like Skype and iCall [1] become widely available for these platforms. In the near future, it may not be unusual for a dozen active WiFi VoIP handsets to be in range of a single WiFi access point, for example at a local Starbucks or at enterprises. As a result, we may increasingly expect WiFi hotspots and enterprise networks to become *mixed-use* environments, where bandwidth is shared by mix of VoIP users and traditional data users, such as Web browsing users.

One might imagine that such a mixed-use scenario would be easily supported by existing installations, as VoIP is a relatively low-bandwidth protocol. For example, given a G.729[2] VoIP codec rate of 6.4 Kbps, with a combined header size of RTP, UDP and IP of 40 bytes, one might expect an 802.11b channel with 11 Mbps of capacity to support over 70 bidirectional VoIP calls and still leave half of the channel capacity for data traffic. It is well known, however, that nothing could be further from the truth; estimates of

[1]We use the terms 802.11 and WiFi interchangeably

[2]In G.729 each direction has a 10-ms inter-packet arrival, an eight-byte voice payload, and twelve additional bytes of RTP header. Variants of G.729 also run at longer inter-packet times and/or increased voice payload sizes.

the number of simultaneous VoIP sessions supported by 802.11b (in the *absence* of data traffic) vary widely, ranging from as few as six to as many as 36 [10, 31, 63], depending upon the particular characteristics of the network and codecs in use, but are a far cry from what a back of the envelope calculation would suggest. This counterintuitive result is due to the large per-packet overhead imposed by WiFi for each VoIP packet—both in terms of protocol headers and due to WiFi contention.

In the mixed-use scenario described, users expect good performance from both VoIP traffic (call quality) and data traffic (throughput). Call quality has traditionally been a major concern for WiFi VoIP deployments, since real-time audio traffic has stringent requirements in terms of loss rate, delay and jitter, and needs to be sent at a high rate (*e.g.*, 50–100 packets per second for many VoIP codecs) to maintain acceptable audio quality. In mixed-use cases, best-effort data traffic can cause excessive queuing of VoIP traffic at access points and may increase packet loss rate due to contention for the medium. Since a VoIP call occupies only a very small amount of bandwidth (possibly as few as eight bytes of voice data per packet), many researchers [10, 81] and commercial providers [2] have proposed prioritizing VoIP packets, with the unstated assumption that the impact on overall network performance will be minimal. However, as we demonstrate experimentally based on a testbed spread out over several offices, as few as six VoIP calls may remove over half of the TCP capacity in 802.11b. Moreover, prioritizing VoIP sessions runs the very real danger of drowning out all competing best-effort traffic, such as Web browsing and email messaging. Somewhat surprisingly, our experiments show that neither the increased speed of 802.11g nor the quality-of-service mechanisms of 802.11e change this reality.

In this chapter, we address the impending potential disaster: that widespread VoIP usage will cripple hotspot and enterprise WiFi networks. In addition to quantifying and explaining the impact of VoIP on the capacity of WiFi, we propose enhancements to 802.11 that effectively aggregate most of the impact of multiple VoIP clients into the equivalent of a single VoIP client, thus reducing VoIP's impact on the network's data-carrying capacity. Given the long cycle of 802.11 standardization (several recent instances have exceeded four years [3, 4]) and current widespread deployment of 802.11 (*e.g.*, one report claiming over 290,000 hotspots worldwide [6]), we specifically target current deployments using a black box approach that leaves 802.11 hardware, such as access points and 802.11 cards in phones and laptops, unmodified. Specifically, we present two complementary techniques, time-division multiple access (TDMA) on uplink traffic (from clients to access point) and aggregation on downlink traffic (from access point to clients). The main challenges in using a black box approach to address the performance of VoIP in 802.11 are the decentralized nature of 802.11 (no single point of control[3]), the fact that 802.11 hardware is manufactured by a diverse set of vendors (harder to find a common API for external control), and the need to control devices using sufficiently fine-grained timing. Previous work has independently proposed techniques similar to uplink TDMA and downlink aggregation but has failed to meet all three criteria for an effective black box approach, in particular protocol-feasibility (by assuming changes to the protocol) and sufficient control (by implementing only coarse-grained timing).

In Section 2.1, we quantify and explain the degradation of data capacity and call quality in a mixed-use 802.11 network, finding two main sources of overhead: framing and contention overhead. Section 2.2 describes our Softspeak system that attacks these two types of overhead using a black box approach. In Section 2.3, we evaluate the performance of Softspeak as well as the performance overhead that can be attributed to using a black box approach. We also evaluate techniques that minimize this overhead. Section 2.4 discusses several limitations of Softspeak, and Section 2.5 summarizes the challenges of using a black box approach to implement Softspeak. Finally, Section 2.6 discusses related work.

2.1 The impact of VoIP on WiFi

In this section we empirically demonstrate the degradation of WiFi data capacity as well as VoIP call quality in the presence of an increasing number of VoIP clients. We then employ a detailed simulation of the 802.11 channel access algorithm to determine the precise source of the problem.

2.1.1 Sources of overhead

The 802.11 protocol is designed to allow clients to access the channel in an uncoordinated manner using an algorithm known as the distributed coordination function (DCF). A station that wants to send a packet must ‘back off’ (delay) a random period of time before attempting transmission and defer when sensing that the channel is occu-

[3]An access point controls only certain aspects of 802.11. In particular, it does not control channel access.

pied by another station's transmission. Unlike Ethernet, 802.11 stations are unable to detect collisions (simultaneous transmissions). Therefore, 802.11 requires that a receiver acknowledges the receipt of a data frame using an ACK frame. A sender infers a loss (whether due to collision or frame corruption) after it fails to receive the ACK frame, in which case it increases its random back off time before attempting retransmission.

Uncoordinated approaches such as DCF are known to be inefficient under heavy load as collisions become more frequent and the total airtime utilization of the wireless channel reduces dramatically due to airtime wasted on garbled frames. This problem is particularly relevant in the case of VoIP traffic, since VoIP clients contend often due to the real-time nature of the traffic. The resulting increased collision rate increases loss and jitter, which in turn degrade TCP performance and harm VoIP call quality.

Furthermore, given the small data payload of VoIP packets the overhead of transmitting the various headers in a VoIP packet becomes considerable: each VoIP packet in a WiFi network is typically encumbered with RTP, UDP, IP, MAC and PHY headers as well as a synchronous 802.11 ACK frame. For example, a G.729 packet may take 157 μs to transmit at the maximum rate in 802.11b, or 273 μs if we include the ACK frame (and assume it is sent at maximum rate). Of this time, the eight bytes of voice data carried inside the packet take up only six microseconds; the entire IP packet requires only 35 μs of airtime, resulting in 680% overhead. Although 802.11g can reduce this overhead to 240% in the best case, the overhead remains substantial at over 400% (again optimistically assuming maximum rates are used) in 'protected mode', which is required when any legacy 802.11b device is present. In protected mode, an 802.11g station must precede each transmission by a low rate clear-to-send (CTS) frame, thus increasing per-frame overhead.

Additionally, airtime usage may increase in response to loss rate, as rate control algorithms frequently lower the transmission rate in response to loss, regardless of whether the loss was due to poor signal quality or frame collision. Finally, we note that the resulting increase in airtime scarcity in turn tends to increase collision probability and loss rate as more stations attempt to seize the channel at once, thereby completing a vicious circle.

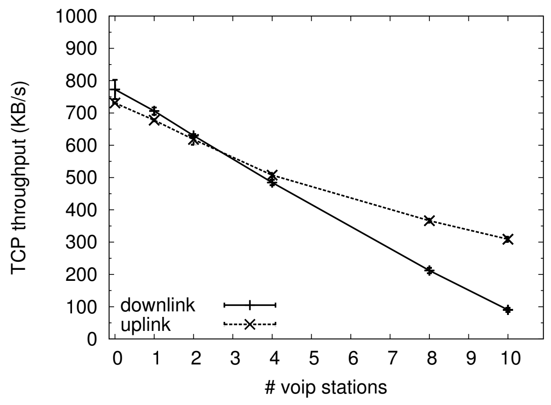


Figure 2.1: TCP throughput as a function of the number of VoIP streams in 802.11b (Avaya AP-8 access point).

2.1.2 Experimental observation

To quantify the impact of VoIP traffic on background data transmissions in current deployments, we have configured a testbed to reflect a realistic scenario for VoIP usage in the enterprise: stations sending and receiving VoIP traffic are spread out over several offices and are connected to an operational building-wide wireless network. For controlled experimentation we ensure that all stations associate to the same access point (AP) and do not roam between different APs. We use wireless cards from two different manufacturers to ensure our results are not artifacts of a particular piece of hardware. (Full details of the testbed are included in Section 2.3.1.)

We begin by considering our default configuration based on 802.11b and voice calls based on a 10-ms G.729 voice codec. We then explore options potentially available in today's deployments that might improve performance. In particular, we study the impact of using a higher MAC rate (802.11g standard), quality-of-service enhancements (802.11e standard), and lower rate voice codecs (20-ms G.729 standard).

Residual capacity

We are interested in the residual WiFi capacity as well as VoIP call quality in the presence of a varying number of VoIP stations. Here, we measure the residual capacity by simultaneously running a bulk flow and measuring its throughput. We conduct separate experiments for uplink and downlink bulk flows, using both TCP and UDP. Our experiments with UDP measure the raw channel capacity available, while

TCP measures the effective capacity for flows that are sensitive to loss and delay. For simplicity, we restrict our discussion to experiments using a single non-VoIP flow at a separate client; we present results for multiple data clients in Section 2.3.5.

Figure 2.1 plots the throughput of TCP in the presence of a varying number of VoIP stations in an 802.11b network. As we increase the number of VoIP streams, the throughput of a TCP uplink flow (where 'uplink' refers to the direction of the TCP data packets) degrades, halving at around eight VoIP streams. In typical TCP usage (*e.g.*, Web traffic) more throughput is required from the downlink direction than from the uplink direction. Unfortunately, throughput degradation is far worse for a TCP downlink flow, which can be explained as follows. TCP's congestion control mechanism attempts to use the maximum bandwidth available given the loss rate and the RTT. For both cases, the TCP sender needs to share the AP with other traffic for its downlink traffic (data packets for TCP downlink or ACK packets for TCP uplink), and it is therefore at the AP that most losses are expected to occur. Losing a data packet is far worse than losing an ACK packet, however. Therefore, TCP is able to tolerate a higher loss rate at the AP and achieve a higher throughput when sending data uplink. As a result, TCP downlink throughput halves at six VoIP streams and degrades by over 85% in the presence of ten VoIP streams.

UDP throughput degradation is less severe than that of TCP because UDP is less sensitive to loss and delay. Nevertheless we observe a significant throughput degradation (over 55% with ten VoIP sessions). We further note that the behavior of uplink UDP and TCP traffic and their impact on VoIP traffic appears quite similar, indicating that in our testbed the TCP uplink behavior is characterized mostly by channel capacity, rather than by loss and delay.

Call quality

As we increase the number of simultaneous VoIP sessions, the individual call quality also decreases. Call quality is a function of packet loss rate, delay and delay jitter, and is typically represented as a Mean Opinion Score (MOS) ranging from 1 (bad) to 5 (good) [5]. We use an approximation of MOS based on network-level metrics [24] with codec-specific parameters calibrated using simulation [25]. We assume a playout buffer that is able to adapt its de-jitter delay such that on average no more than 1% of packets are late. We find that in the presence of TCP and bulk UDP uplink traffic, MOS decreases from 3.8 to 1 as the number of VoIP stations increases from one to ten. In these cases VoIP traffic undergoes severe loss (reaching 50%) due to drop-tail queuing at the AP queue where it competes with bulk data or TCP acknowledgments. Conversely, TCP downlink traffic is suppressed by VoIP traffic to such an extent that the VoIP MOS remains relatively unaffected. A major challenge is thus to improve TCP downlink performance without sacrificing call VoIP quality.

802.11 protocol extensions

To evaluate whether higher bit rates alleviate problems of contention and overhead we perform the same set of experiments using 802.11g. We find that throughput degradation is less severe in pure 802.11g networks than in 802.11b. For example, TCP downlink performance does not drop as sharply as it does in 802.11b, but degrades in a similar way to TCP uplink and UDP performance. The loss in capacity when ten VoIP clients are present is still substantial, however, ranging from a 32% reduction in the case of UDP downlink to 39% for TCP downlink traffic. Similarly, while VoIP MOS is higher in 802.11g, it is still unacceptably low, dropping from 3.8 to 1.3 as the number of VoIP sessions increases from one to ten due to frequent losses.

In practice, however, our enterprise WiFi deployment almost never supports only 802.11g clients, forcing 802.11g stations to use protected mode whenever a 802.11b station is detected. We observe that the capacity degradation caused by 802.11g VoIP clients in an 802.11g protected-mode network is comparable to that of native 802.11b. Thus, the presence of a single legacy 802.11b client (VoIP or otherwise) alongside ten VoIP clients removes 87% of TCP downlink capacity. In addition, we find that whereas VoIP uplink loss is negligible in 802.11b in the presence of TCP downlink traffic, it varies from 10–40% in 802.11g protected mode, resulting in an average VoIP MOS value of 2.0.

In 2005, IEEE standardized 802.11e, a set of quality-of-service enhancements to 802.11 specifically designed to allow real-time and data traffic to co-exist efficiently. 802.11e introduces two channel access mechanisms, EDCA and HCCA. HCCA, in which the AP explicitly arbitrates channel access, is designated optional and rarely implemented. EDCA is an enhancement of 802.11's DCF channel access mechanism, and includes the ability for a station to prioritize VoIP traffic by reducing the backoff time before it sends a VoIP packet. In the remainder, we use the term 802.11e to indicate this prioritization mechanism. We compare the performance of 802.11b and 802.11b+e using

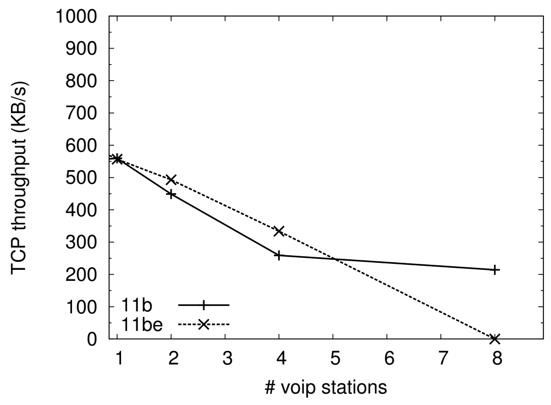


Figure 2.2: TCP uplink throughput as a function of the number of VoIP stations in both 802.11b and 802.11b+e (Linksys WAP4400N access point).

a popular 802.11e-capable access point (a Linksys WAP4400N, different from the Avaya AP-8 used in the previous experiments, which does not support 802.11e), with VoIP traffic configured to be classified and prioritized over other traffic at both the AP and the clients. In the presence of TCP uplink traffic, we observe that compared to 802.11b, 802.11e does indeed improve the MOS of VoIP traffic. However, as shown in Figure 2.2, this improvement is achieved at the expense of TCP uplink throughput, which degrades far more severely than is the case for 802.11b. TCP downlink performance is essentially similar to that of 802.11b, with a slight improvement in MOS. We conclude that while 802.11e (at least as implemented by a popular AP vendor) is able to improve call quality in some cases, it does not mitigate throughput degradation in the presence of a large number of VoIP clients.

Less aggressive voice codecs

By combining multiple 10-ms voice frames into a single IP packet, G.729 can be run at longer inter-packet intervals, thereby making more efficient use of network resources. Figure 2.3 considers a 20-ms G.729 codec in combination with TCP in 802.11g protected mode. As expected, the impact is less than for a 10-ms codec yet remains severe; the MOS for uplink VoIP traffic drops from 4 to 3 on average (compared to 2 in the 10-ms case) and, more importantly, becomes highly erratic. Uplink and downlink TCP throughput reduce by around 40% (*cf.* 87% in the 10-ms case for TCP downlink).

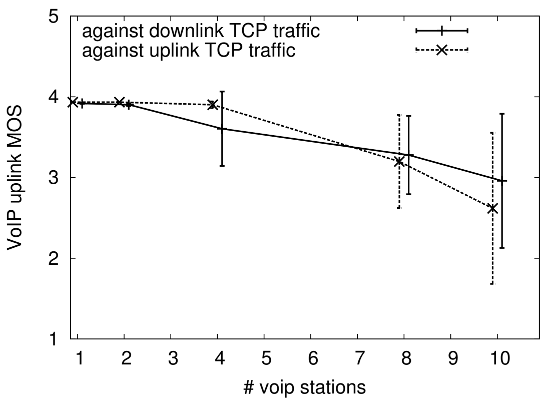


Figure 2.3: Uplink MOS of a 20-ms codec in 802.11g (protected mode) in the presence of TCP traffic. (Data points are slightly offset to avoid overlapping error bars.)

2.1.3 802.11b simulator

While our experiments clearly demonstrate real-world performance problems, it is often difficult to determine to what extent the degradation measured is due to the 802.11 protocol rather than interference, fading, hidden terminals, or other environmental factors. In order to cleanly separate these factors, we have implemented an 802.11 protocol simulator that allows us to evaluate how aspects of the standard distributed coordination function (DCF) algorithm impact performance, in particular residual capacity. We specifically omit the simulation of RF properties, rate adaptation, background broadcast traffic (*e.g.*, DHCP and ARP), and hardware imperfections, in order to show that the DCF algorithm by itself explains our experimental observations of residual capacity. We focus on the percentage of time a client uses the medium, since it not only directly reflects bulk UDP throughput, but also indirectly reflects loss rate: in a DCF-based model losses are caused by colliding packets, which in turn occupy airtime.

Configuration and validation

The simulator contains objects representing the AP and wired and wireless stations that send UDP traffic (bulk traffic or based on the traffic characteristics of a VoIP codec). Wired stations are modeled as directly connected to the AP. The wireless stations and AP contend for access using the standard 802.11 DCF algorithm. We parameterize the simulator to mimic the behavior of our testbed hardware (particular settings are detailed later in Table 2.1) and use a bit rate of 11 Mbps. We configure an AP queue

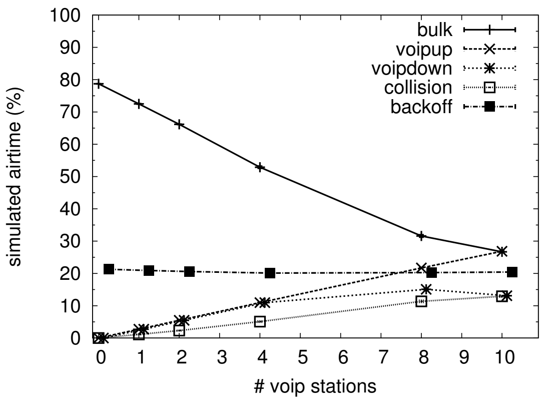


Figure 2.4: Simulated airtime versus the number of VoIP streams, in the presence of 802.11b UDP uplink traffic.

length of 500 and station queue lengths of 10, but note that our simulation results are not sensitive to the choice of queue-length parameters.

We simulate the 802.11b experiment described earlier for UDP and find that the results are very similar in airtime. For example, simulated throughput degradation is within 10% of the experimental results. The largest difference between the simulated and experimental results is seen in the uplink VoIP loss rate which is 0.8–2.3% for ten VoIP stations versus less than 0.02% on the testbed.

DCF's share of VoIP impact

Having established that our simulation exhibits a similar behavior as the testbed in 802.11b, and that a DCF-based model is sufficient to explain the degradation of residual capacity in our testbed under VoIP, we now analyze the simulation data to determine which aspect of DCF causes the observed behavior. Figure 2.4 shows the simulated airtime used by each of the following components: non-colliding bulk traffic (*bulk*), non-colliding VoIP uplink and downlink traffic (*voipup*, *voipdown*), colliding packets (*collisions*), and times when all stations are backing off or sensing the medium (*backoff*).

VoIP takes up a large fraction of the airtime, *e.g.*, 40% for ten sessions, exceeding the airtime used by bulk traffic. Most of the VoIP airtime (35%) consists of framing overhead. Additionally, 33% of total airtime is overhead due to contention (20% backoff plus 13% wasted on collisions). The techniques presented in the next section are capable of reducing a significant portion of overhead, specifically the framing overhead of downlink VoIP traffic (11%) and the collision time (13%). Based upon these numbers alone there is potential to almost double the residual channel capacity.

2.2 Design and implementation of Softspeak

Softspeak targets the key challenges of excessive contention and framing to build a software-only solution that can be deployed on existing commodity hardware. The main idea is to aggregate voice traffic by combining many small packets into larger ones, thereby reducing per packet overhead. Others have observed that all *downlink* packets must pass through the AP; hence, the opportunity to aggregate exists at either at the AP itself or just before the packets are sent to the AP [75, 76]. However, physically aggregating *uplink* VoIP packets is challenging since there are multiple, independent VoIP senders. Instead, we propose a time-division multiple access (TDMA) scheme that approximates uplink aggregation to the extent that it provides a similar reduction in contention overhead. Our uplink TDMA scheme can function independently of the downlink scheme and requires only client-side modifications. Downlink aggregation, on the other hand, also requires either modifying the AP, or, more realistically, adding a separate 'VoIP aggregator' device upstream from the AP. Using a black box approach, both mechanisms conform to the existing 802.11 specification (protocol-feasible), coexisting with VoIP stations that do not use Softspeak.

2.2.1 Uplink TDMA

Our uplink approach reduces the amount of contention created by VoIP clients. Specifically, we alter the contention behavior of the VoIP clients to no longer contend with non-VoIP clients, and then devise a distributed mechanism to schedule the VoIP clients in a TDMA fashion so that they no longer contend with each other either.

We remove the VoIP clients from the standard contention process by modifying their backoff behavior. Instead of sensing the medium for the 802.11-mandated DCF inter-frame spacing (DIFS) followed by a random backoff before sending, a Softspeak VoIP client senses for a shorter period of time and does not perform backoff, thus preventing collisions with non-VoIP traffic. (In the absence of hidden terminals, collisions with ACKs are prevented by 802.11's NAV mechanism.) This behavior effectively prioritizes uplink VoIP traffic and improves call quality. (A similar mechanism is employed by a commercial product, SVP [2].) By itself, however, this alteration inhibits DCF's ability to prevent collisions among the VoIP stations. In fact, when we simulate only two VoIP stations that sense for a short inter-frame spacing (SIFS) without backoff in combination with bulk traffic that uses standard contention, we find that neither VoIP station is able to sustain a viable VoIP session.

To prevent VoIP stations from colliding with each other, we introduce coarse-grained time slots and construct a TDMA schedule for the VoIP clients. When used in combination with downlink aggregation, the downlink aggregator node can assign TDMA slots as well as perform admission control, since it has knowledge of all the clients using our scheme. In the absence of a centralized scheduler, we devise a distributed mechanism (Section 2.2.1) that leverages management frames within the 802.11 protocol to allocate slots.

Slot allocation and admission control

In an ideal deployment, the network operator will have installed a Softspeak VoIP downlink aggregator that can assign slots for uplink TDMA. If all available slots are in use it can deny access to a new Softspeak client, in which case that client resorts to normal 802.11 DCF. In some scenarios, however, it may be easier for individual clients to install Softspeak software than to convince network operators to install new hardware. Moreover, uplink TDMA is useful by itself, *i.e.*, without downlink aggregation, since it reduces contention by uplink VoIP stations. Hence, if clients are unable to locate a VoIP aggregator (Section 2.2.2 describes the registration process), they proceed with a distributed allocation process.

Independent of how TDMA slots are allocated to clients, VoIP stations need to be synchronized in order to correctly use their assigned slots. Each client uses the periodic beacon frame broadcast by an 802.11 AP to synchronize with other VoIP clients. Beacons are sent at fixed intervals (usually 100 ms), and, since they are sent by the AP at a low bit rate, are typically received by all clients. It is important to note that a VoIP client may also hear beacons from an AP other than the one to which it is associated. To use beacon-based synchronization, VoIP clients need two important pieces of information: a) The AP to whose beacons other nearby VoIP clients are synchronizing, and b) which TDMA slots they are using. The slot allocation process provides both pieces of information. In the case of distributed slot allocation each VoIP client encodes the information by temporarily spoofing its MAC address (6 octets) as follows:

- The first three octets (known as the OUI) are taken from a reserved OUI address space to ensure the resulting address is valid and unique.
- The next two octets are the same as the *last two* octets of the BSSID of the AP to whose beacons the VoIP station is synchronizing.
- The last octet is used to denote the particular real time slot the VoIP station is using or wants to use.

The main concern when coordinating clients is that there is no guarantee they can hear each other's transmissions. Hence, Softspeak clients coerce the AP into generating specially crafted packets that the other clients can hear. VoIP stations using uplink TDMA periodically (*e.g.*, once a second) send directed Probe-Requests on the channel and to the AP to which they are currently associated using the modified MAC address. The destination (unmodified) AP will respond with a Probe-Response packet whose destination is the VoIP station's modified MAC address, which is heard by all associated clients.

A new VoIP station that wants to use uplink TDMA first enters promiscuous mode for a few seconds to sense the channel to check if there are any special Probe-Response packets (easily identifiable by the first three octets of the destination MAC address), thus determining which AP's beacons are being used for synchronization and which slots are in use. If the VoIP client detects any such Probe-Responses, it extracts the encoded AP and uses that for TDMA synchronization. Otherwise it synchronizes using the AP with which it is associated. In either case, the VoIP client picks an unused slot and starts to periodically broadcast a Probe-Request with its source MAC address denoting its slot and the AP it is using for synchronization. As before, the AP sends Probe-Responses which can be heard by new VoIP clients wanting to join. Finally, when a VoIP station finishes its session it stops sending Probe-Requests.

Our slot assignment scheme seamlessly supports dynamic node arrivals and departures. Moreover, this scheme works even when nearby clients are associated to different APs, since a client may synchronize with an AP other than the one it is associated to. Finally, our scheme works if APs use various 802.11 security features since Probe-Request and Probe-Responses are always sent unencrypted. We have deployed our scheme with an AP that employs MAC-address-based access control, WPA2 or WEP encryption, and disabled SSID broadcasting.

A drawback of the distributed allocation scheme as currently described is that it is unable to detect multiple clients attempting to allocate the same slot simultaneously. We observe that this problem can be solved (or made unlikely to occur) by adding some bits of randomness to the spoofed MAC address, allowing the clients to arbitrate among conflicting slot allocations. For example, the scheme may be extended by having VoIP clients announce the BSSID and the slot number in separate Probes, thus allowing room for some bytes to be set randomly by each client.

Synchronizing TDMA slots

To implement uplink TDMA, we modify the Ralink RT2560F wireless card protocol stack in Linux 2.6.21 (without modifying the WiFi hardware or firmware). Ideally, once slots are allocated, each VoIP station contends for the channel in its assigned slot and refrains from contending outside its slot. By default, the Linux 2.6 kernel timer interrupt is programmed to fire every millisecond; we show later that this also happens to be close to the optimal granularity for VoIP slotting in 802.11b. Using one-millisecond slots, a TDMA scheme can support ten simultaneous VoIP stations using a codec with 10-ms inter-packet arrival rate, or 20 stations using a 20-ms codec. Since 802.11a/g frames for these codecs take less airtime, Softspeak could use smaller slots, allowing a larger number of VoIP stations to be admitted; we have not yet implemented sub-millisecond slotting.

A straightforward implementation of one-millisecond slotting is to suspend and resume transmission from within Linux's timer interrupt handler in accordance with a station's assigned slot. However, the naïve approach faces two problems: clock skew and timer inaccuracy. Figure 2.5 illustrates both. In this experiment, a single station uses `iperf` to emulate a G.729 VoIP codec with a 10-ms inter-packet arrival rate. We manually assign the station a static TDMA slot; there is little to no background traffic on the same AP during the experiment

In the figure, the x axis plots time in seconds, and the y axis shows the start time of each transmission modulo 10,000 μs (10 ms). The figure shows the effect of the timer interrupt firing faster than 1,000 times per second as well as `iperf` sending slightly slower than the configured rate of 100 packets per second. If the timer interrupt and

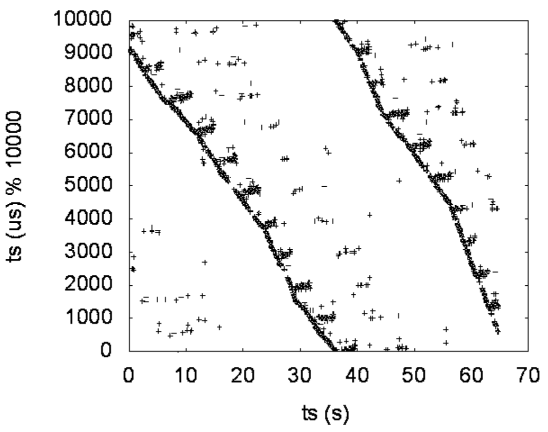


Figure 2.5: Time series of transmission times by a single station, no synchronization.

`iperf` operated at their correct rate, we would expect to see a single horizontal band corresponding to the station's assigned slot. Instead, `iperf` schedules packets at a rate slower than the timer interrupt, and as a result iperf and the implemented TDMA slot drift with respect to each other. When `iperf` happens to send inside the slot, a short almost horizontal line appears starting at the bottom of the slot (the slight upward slope of this line is the clock skew). Once transmissions reach the top of the slot, packets are buffered until the start of the next slot, causing the downward sloping lines. The slope is caused by the timer interrupt firing too fast.

Different stations may exhibit different degrees of skew, possibly even varying across time. We address this issue by effectively slaving each station's clock to an AP. Specifically, we reset the timer every time a station hears the periodic beacon frame from the AP that was assigned during the slot allocation process. On the Soekris net4801 in our testbed, Linux uses the programmable interval timer (PIT) as its time interrupt source. Therefore, we modify the driver to reset the PIT every time it hears a beacon, which we have measured to be roughly once every 102–103 ms for the APs in our network.

Manipulating the PIT timer in this way may conceivably cause unintended timing artifacts in the station's operation. Therefore, we have developed an alternative implementation that uses Linux's high-resolution timers to schedule the VoIP slots and have observed a similar degree of synchronization. However, the results in this dissertation are based on manipulating the PIT timer.

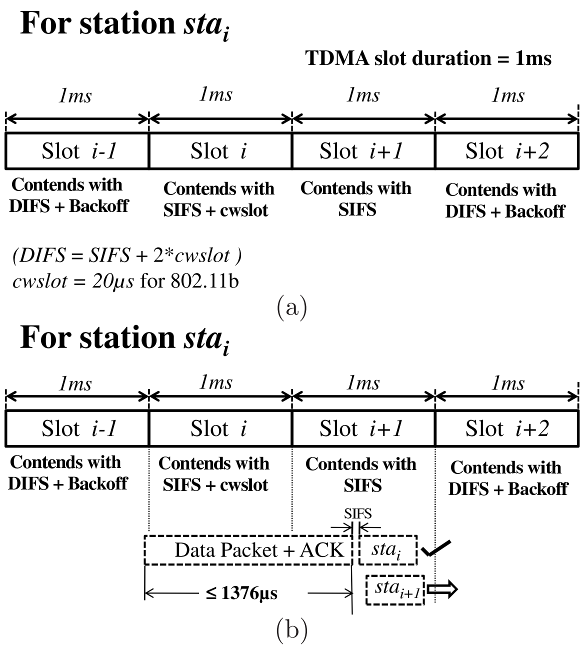


Figure 2.6: Illustration of dynamic IFS. (a) The various contention parameters used by dynamic IFS as dependent on the TDMA slot sta_i is contending in. (b) Dynamic IFS in the presence of other data traffic: in TDMA slot $i+1$ sta_i wins over sta_{i+1} since it contends with SIFS rather than SIFS + cwslot.

Controlling transmission timing

An obvious complication with our scheme is that when a TDMA slot starts, a station other than the station that has been assigned the slot may already be transmitting a frame. At 11 Mbps a maximum-sized IP packet (1500 bytes) together with ACK will take 1376 μs, potentially delaying the station by that time from the start of its slot into the next slot.[4] In addition, the VoIP station may repeatedly fail to capture the channel even while actively contending. We address this challenge by letting the WiFi card driver adjust the way VoIP station contends for the channel during its assigned slot, a mechanism we term *dynamic IFS* (dynamic inter-frame spacing).

In standard DCF, stations contend using an inter-frame spacing of SIFS + $(2 \cdot$ cwslot) followed by a random backoff. (By cwslot we denote an 802.11 contention-window slot—20 μs in 802.11b—not Softspeak's 1-ms TDMA slot.) We use the two 20-μs cwslot intervals starting at SIFS and (SIFS + cwslot), respectively, to (a) prioritize the VoIP traffic over non-VoIP traffic and (b) prioritize among different VoIP stations to avoid

[4]We assume short preambles throughout this chapter.

collisions. Accordingly, we let each station contend as follows: Figure 2.6(a) considers a station sta_i which is assigned TDMA slot i. During the station's assigned TDMA slot it contends with (SIFS + cwslot) (and no backoff). In slot $i + 1$, it contends with SIFS (and no backoff). In any other slot it contends as specified by DCF (SIFS + (2 · cwslot) + backoff).

Now let us consider the scenario as illustrated in Figure 2.6(b), in which a station sta_i in TDMA slot i is delayed into the next TDMA slot (i+1) by an ongoing transmission and assume for the moment that sta_i's packet was ready at the start of the slot i. After the transmission has ended, stations sta_i and sta_{i+1} contend for the channel. However, due to the assigned contention parameters, sta_i is guaranteed to win over station sta_{i+1}. Furthermore, after sta_i has finished transmitting and received its ACK (after 430 μs for a large-payload G.711 codec), there is still at least (2 ms - 1376 μs - 430 μs = 194 μs) for sta_{i+1} to commence its transmission and therefore not contend in TDMA slot $(i+2)$. It can be shown that in the absence of retransmissions, as long as (a) the duration of a VoIP frame is less than one TDMA slot and (b) the duration of a bulk frame is less than two TDMA slots, station i will never contend in slot $(i + 2)$. Even if due to, *e.g.*, 802.11 retransmissions or imperfect control of timing by Softspeak, a station ends up contending in a TDMA slot other than i or $(i+1)$, it will do so using conventional DCF contention parameters and do no worse than without our improvements.

Figure 2.7 plots the transmission start times of ten VoIP stations, each assigned a separate TDMA slot, when competing against background traffic. In particular, a bulk UDP sender generates background traffic in the downlink direction to a separate wireless station. Using dynamic IFS, the slotting is clearly defined: while the bands are longer than 1 ms due to delays caused by ongoing background traffic transmissions (as explained above), the majority of transmissions do not commence more than one slot away.

The first slot (assigned to the VoIP station plotted in the first column of Figure 2.7) commences roughly 500 μs after the beacon time. This offset is caused by inevitable delays between the time that the beacon is generated by the AP and when it is received and processed by a station, and also between the time the station driver generates a packet for a particular slot and the time that it is transmitted. In particular, 400 μs of this time is accounted for by beacon transmission time, the remainder consisting of processing delays in the station. While some of these processing delays

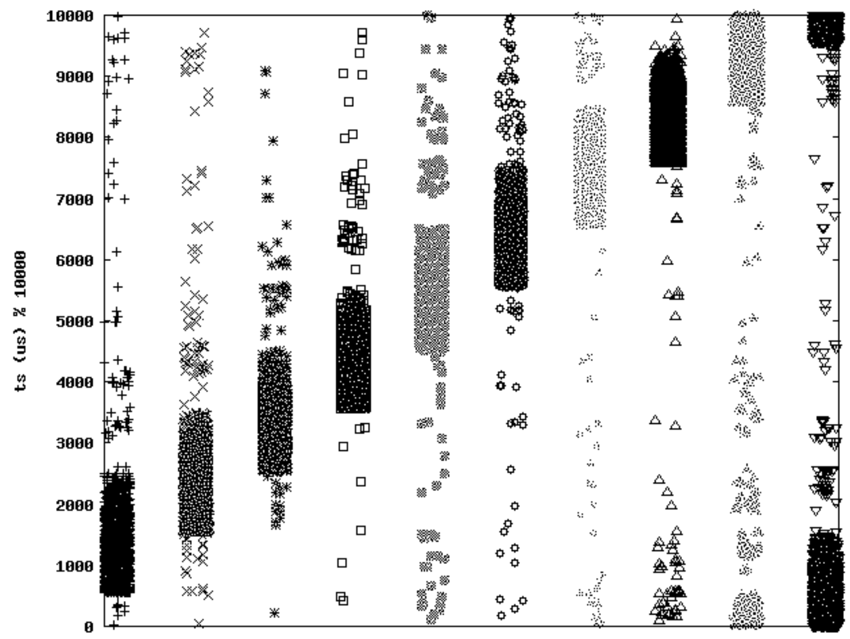


Figure 2.7: TDMA slotting by ten VoIP stations using dynamic IFS in the presence of UDP downlink background traffic. Each column represents a distinct VoIP station.

may vary across different stations, as subsequent figures show, the delay is consistent enough across multiple stations with the same hardware configuration that a station's synchronization can be tuned for that hardware.

2.2.2 Downlink aggregation

Downlink aggregation introduces an aggregator component that is placed at or before the WiFi AP (uplink from the AP). The aggregator is on-path and transparently forwards all traffic to and from the AP; non-VoIP traffic is forwarded without modification. The aggregator buffers VoIP frames destined for wireless stations and releases a frame encapsulating the buffered frames at a regular interval (every M ms, where M is the minimum packetization interval of the VoIP codecs in use.) By combining all the VoIP sessions into one packet per codec interval, downlink aggregation can virtually eliminate the marginal header and contention overhead of additional VoIP clients. There is a down side however: when the aggregator buffers a packet, it adds a constant delay of $M/2$ ms in expectation, *e.g.*, 5 ms given a 10-ms codec.

When a new Softspeak VoIP session starts up (or when the station roams to a different AP) it registers with the aggregator node, which we implement on a separate Linux machine. When the aggregator receives a downlink packet addressed to a registered VoIP client, it buffers the packet and combines it with all other buffered packets into a single encapsulated packet that it sends out at fixed intervals (*e.g.*, 10 ms for G.729). The aggregator node uses the IP header information from the most recently heard uplink packet (say from station $S1$) to construct a new frame. Addressing the packet to $S1$ increases the likelihood that the packet will be acknowledged by a currently active VoIP client. We define an aggregation header that stores the set of destinations and original IP packet lengths for each station. The aggregation header is prepended to the UDP header and packet payload for $S1$, and then the respective IP and UDP headers and payloads for the remaining buffered VoIP packets are appended.

In contrast to previous proposals [75], we address the aggregated frame to only one of the VoIP stations; we configure the WiFi interface of each of the VoIP stations to be in promiscuous mode to allow them to receive the aggregated packets regardless of the destination. The client passes aggregated packets to the Softspeak module that de-encapsulates the packet, extracts the portion meant for the current station, and passes it up the networking stack. Because the aggregated packet is addressed to only one station, there will be at most one MAC-layer acknowledgment. Wang *et al.*, on the other hand, propose the use of multicast in order to eliminate the MAC ACK frame. We preserve the ACK frame for two pragmatic reasons. First, in our experience, while obviously unable to eliminate all loss, the single ACK frame is a cost-effective mechanism to protect the aggregated packet against many collisions. Secondly, and perhaps more importantly, commodity access points typically transmit multicast frames only at a multiple of the beacon interval to interoperate with clients in power-save mode, introducing intolerable delay. In terms of our criteria for an effective black box approach, we claim that such previous proposals are not protocol-feasible for VoIP. We defer a full discussion of the black box aspects of Softspeak to Section 2.5.

2.3 Evaluation

We now evaluate the effect of downlink aggregation and uplink TDMA, both independently and in concert. First, we show that our schemes significantly increase the available channel capacity while usually maintaining—and sometimes improving—VoIP call quality. Second, we evaluate the overhead of using a black box approach to implement

Table 2.1: 802.11b contention parameters measured for our wireless hardware.

Card	CWmin	CWmax	Retry limit
Ralink RT2560F	8	256	8
Atheros AR5212	32	32	11
Avaya AP-8	16	16	11

Softspeak and find that it is close to optimal in terms of throughput improvement.

2.3.1 Experimental testbed

The wireless infrastructure in our building is a managed 802.11b/g deployment of enterprise-class Avaya AP-8 access points. There are multiple APs per floor which are configured to orthogonal channels to increase spatial diversity. We configure eleven Soekris net4801 boxes to act as VoIP stations. Each has two mini-PCI wireless cards: an Atheros AR5212 chipset-based card and an Ralink RT2560F-based interface. The net4801 is a single-board based computer with a 266-MHz CPU running the Linux operating system. To simplify our experiments, we emulate VoIP traffic using `iperf`. We use `iperf` to generate UDP traffic that mimics a commonly used VoIP codec, G.729, at 10-ms inter-packet intervals. RTS/CTS is disabled on all Soekris boxes and APs. All experiments are conducted late at night to minimize background wireless activity.

We employ ten commodity PCs connected over wired gigabit Ethernet as end-points for the (emulated) VoIP traffic generated by the Soekris boxes. Essentially, each PC-Soekris pair serves as a distinct bi-directional VoIP call. One additional PC-Soekris pair conducts a bulk transfer (TCP or UDP) to measure the residual capacity of the wireless channel in the presence of the VoIP traffic. The TCP receive-window size is configured to be large enough that our TCP transfers are never receive-window limited. Unless otherwise noted, bulk transfer is conducted through the Atheros card, while the Ralink interfaces send and receive VoIP traffic.

Table 2.1 reports the default contention parameters for the various devices in our testbed as measured by the Jigsaw wireless monitoring infrastructure [21]. We note that neither the Atheros card nor the Avaya AP appears to double its contention window size on retries, in contrast with the default behavior specified by 802.11.

2.3.2 Results for 802.11b

Figures 2.8 and 2.9 compare bulk throughput and VoIP call quality across all combinations of applying uplink TDMA and/or downlink aggregation in 802.11b, for TCP uplink (Figure 2.8) and downlink (Figure 2.9). The results for UDP bulk uplink are similar to those of TCP uplink. We discuss the case of UDP bulk downlink in Section 2.3.3. The most important conclusions are that (a) applying a combination of uplink TDMA and downlink aggregation improves residual bulk throughput, in some cases drastically, (b) with one exception, call quality is preserved or greatly improved, (c) applying only one of uplink TDMA or downlink aggregation does not achieve these results across all three cases of bulk traffic load.

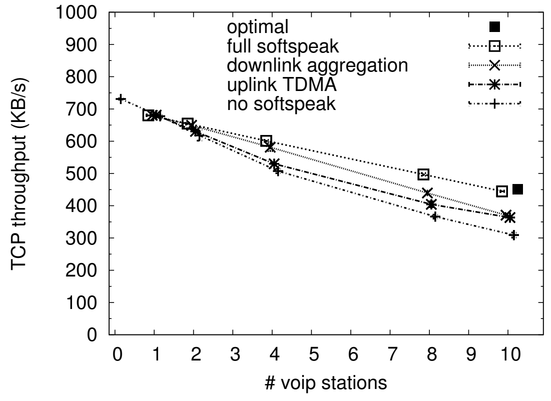
We summarize the benefits of Softspeak (combined uplink TDMA and downlink aggregation) over 802.11, for the case of ten VoIP sessions, as follows:

TCP uplink and UDP uplink: Capacity increases by around 50% (Figure 2.8(a)). Downlink VoIP improves from being completely unusable for VoIP to being usable (Figure 2.8(b)). The bulk of this improvement comes from a reduction in downlink loss rate (from 55% to 4.8%) by downlink aggregation. However, uplink TDMA contributes significantly by further reducing the downlink loss rate (to 1.8%), resulting in a substantial increase in MOS. For uplink VoIP (Figure 2.8(c)) most of the MOS improvement comes from downlink aggregation, which reduces the RTT from over 400 ms to below 25 ms by reducing queuing at the AP.[5]

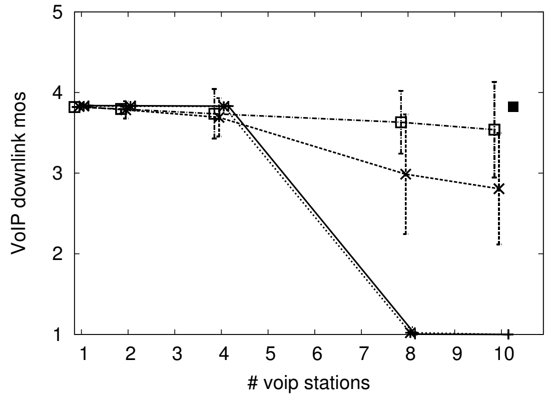
TCP downlink: Capacity multiplies 4.8 times (380% increase) from 92 KB/s to 445 KB/s (Figure 2.9(a)). Unfortunately, VoIP downlink MOS degrades somewhat (Figure 2.9(b)). On closer examination, we find that downlink MOS suffers from an increased loss rate from downlink aggregated packets: since Softspeak's downlink aggregation scheme receives link-layer acknowledgments from only one VoIP client, only frame losses experienced by that client result in retransmission. Frame corruption experienced by other clients remains unnoticed. We address this issue when we present our results for 802.11g (Section 2.3.4) where higher frame rates may further increase the probability of frame corruption.

While these results show that Softspeak improves the efficiency of 802.11b networks in the presence of VoIP in terms of residual TCP capacity (while mostly preserving

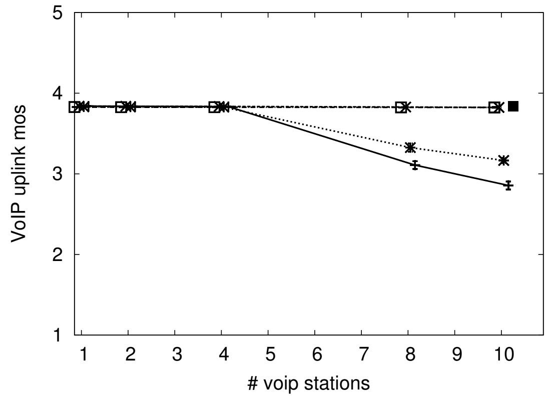
[5]Note that delay in one direction affects MOS in both directions.



(a) Capacity (uplink TCP)

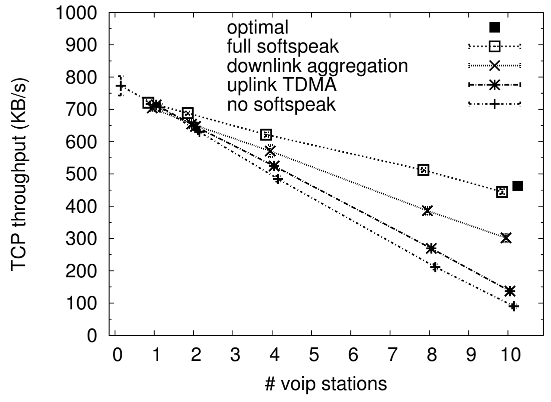


(b) Downlink MOS (uplink TCP)

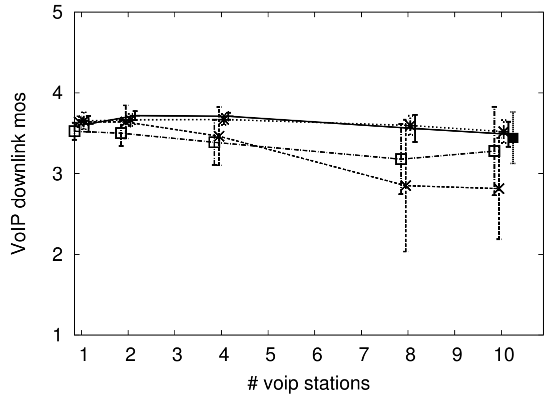


(c) Uplink MOS (uplink TCP)

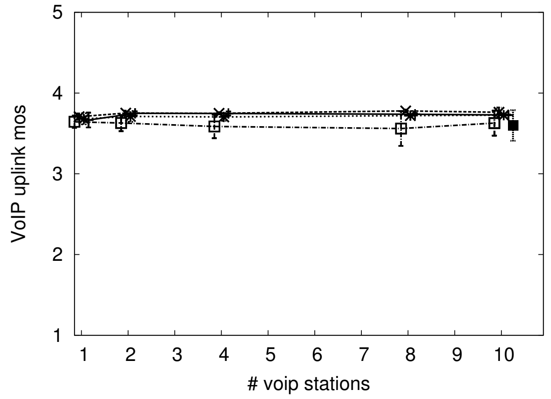
Figure 2.8: Impact of Softspeak on a varying number of VoIP stations in combination with TCP uplink traffic (802.11b). Softspeak increases capacity by around 50% and improves average call quality from MOS 1 to close to the maximum observed across experiments. Softspeak performance approaches that of an emulation of an 'optimal' implementation on average, but increases standard deviation of downlink MOS.



(a) Capacity (downlink TCP)



(b) Downlink MOS (downlink TCP)



(c) Uplink MOS (downlink TCP)

Figure 2.9: Impact of Softspeak on a varying number of VoIP stations in combination with TCP downlink traffic (802.11b). Softspeak increases capacity 4.8 times at the expense of a slight decrease in call quality. However, Softspeak performance still approaches that of our emulation of an 'optimal' implementation, which also decreases call quality.

VoIP call quality), an important question is whether our results are nevertheless limited by our use of a black box approach. For example, it might be the case that our implementation of uplink TDMA lacks sufficient control of VoIP packet scheduling, causing collisions. An 'optimal', clean slate implementation (*e.g.*, one that is implemented in the 802.11 hardware or firmware) might do a better job at controlling the emission of frames according to the TDMA schedule.

To investigate to what extent further improvements may be made by following a clean slate approach (while still remaining compatible with 802.11), we compare our results with those based on an emulation of an optimal implementation. We emulate downlink aggregation by replacing the individual VoIP senders that generate downlink VoIP traffic by a *single* sender that generates packets of the size produced by the downlink aggregator, eliminating any jitter and loss potentially caused by the downlink aggregator. Furthermore, downlink packets are sent to, and their loss rate measured at, a single VoIP station, eliminating any losses due to imperfect overhearing. We emulate uplink TDMA by replacing the VoIP stations by a single VoIP station that sends packets on behalf of all VoIP stations, in other words, it sends packets at ten times the codec rate. The single VoIP station naturally serializes the transmission of uplink VoIP packets, thereby eliminating any collision among VoIP stations. To minimize the probability of colliding with other traffic, it uses SIFS without backoff. In Figures 2.8 and 2.9 the results of the emulation are plotted as an 'optimal' point for ten VoIP clients. In terms of capacity and uplink MOS, Softspeak achieves close to what is optimally achievable. For downlink MOS, consistent with our earlier observation, Softspeak performs worse than optimal due to imperfect overhearing. However, note that in Figure 2.9(b) even optimal Softspeak's downlink MOS is worse than that of 'no softspeak'. This may be expected, given that (optimal) Softspeak enables TCP traffic to considerably increase network resource usage. For example, we measure a 25% increase in RTT (as well as an increased RTT variance) due to a higher AP queue occupation, which in turn explains the higher loss rate of downlink VoIP traffic.

2.3.3 UDP and 802.11e

While Softspeak can improve the capacity available for bulk UDP downlink traffic in 802.11b networks (Table 2.2), it cannot simultaneously reduce the high VoIP downlink loss rate that result from competing with a CBR UDP flow. These losses are caused

Table 2.2: The effectiveness of combining Softspeak (Spk) with prioritization (Prio) in the presence of ten VoIP stations and downlink bulk UDP traffic (802.11b, simulated). (UDP throughput without VoIP is 924 KB/s.)

Metric	No Spk	Spk	Spk+Prio
Downlink bulk tput (KB/s)	375	605	561
Downlink VoIP loss rate	67%	61%	<0.1%
Uplink VoIP loss rate	0.82%	<0.1%	<0.1%

by the AP queue filling with bulk UDP downlink traffic, combined with the fact that UDP does not respond to increasing loss and delay. Similarly, when replacing a single bulk TCP stream by a sufficiently large number of bulk TCP streams, the AP queue fills up with TCP packets causing large delay. These losses and delays can only be ameliorated by adding prioritization at the AP: (aggregated) VoIP packets would therefore not be dropped regardless of the amount of non-VoIP traffic buffered at the AP. Luckily, prioritization is part of the 802.11e standard.

Prioritization

Unfortunately, our testbed hardware cannot simultaneously support Softspeak (which is currently only implemented for the Ralink interfaces) and 802.11e (supported only by the Atheros chipset). We therefore evaluate Softspeak combined with 802.11e-like prioritization at the AP using our simulator. Consistent with our results in Section 2.1.3, our simulator produces results similar to those measured experimentally for the case of UDP without prioritization for the combination of uplink TDMA and downlink aggregation, and we therefore believe that we can extrapolate to the case of AP prioritization. Table 2.2 shows that when we combine Softspeak with prioritization, we not only achieve a 47% improvement on downlink bulk UDP capacity, but also improve VoIP loss rate compared to the baseline.

Airtime utilization

Implementing Softspeak in our simulator also allows us to isolate the source of our performance improvement. Figure 2.10 shows the simulated airtime plot corresponding to Figure 2.4, but with uplink TDMA and downlink aggregation enabled (and no

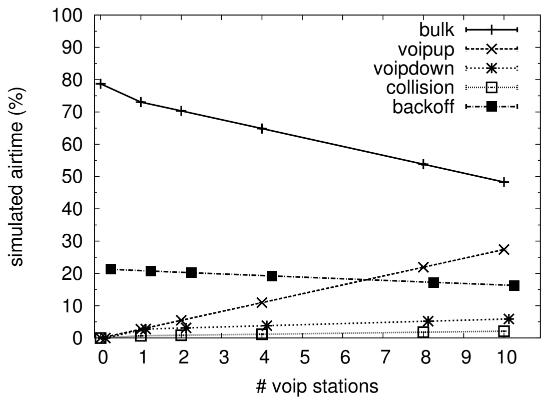


Figure 2.10: Simulated Softspeak airtime usage versus the number of active VoIP streams, in the presence of 802.11b UDP uplink bulk traffic (*cf.* Figure 2.4).

prioritization). The figure indicates that we have achieved our objective of converting almost all time spent on downlink framing overhead and on collision into bulk data capacity. Consistent with the reduction in collision airtime we have also reduced the collision rate, thereby improving loss rate, jitter, and as a result, VoIP call quality and TCP throughput.

2.3.4 Results for 802.11g

For 802.11g we observe that Softspeak as currently described makes significant improvements in capacity (24–32% for ten VoIP stations), while maintaining or lowering jitter and VoIP uplink loss to negligible levels. Recall that when 802.11g runs in protected mode, TCP downlink capacity suffers tremendously in the presence of VoIP. Using Softspeak we are able to triple (increase by 200%) the TCP downlink capacity for ten VoIP stations. However, Softspeak also introduces significant downlink VoIP loss, rising to 30% for some stations, where in some cases virtually none was experienced without enabling Softspeak. In the case of 802.11g protected mode this reduces MOS from 3.7 to 2.8 on average and substantially increases the variance of MOS (Table 2.3, *no measures*).

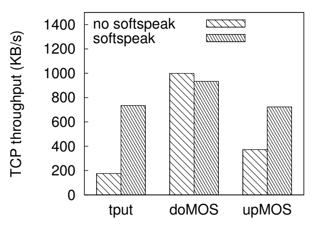
As noted in Section 2.3.2 for 802.11b, downlink aggregation is susceptible to frame corruption by any receiver that is not the link-layer recipient of the aggregated packet, and the higher rates of 802.11g only increase the likelihood of frame corruption. Our solution to this problem is three-fold. First, we observe that judiciously selecting a fixed

Table 2.3: Downlink aggregation losses in the presence of TCP downlink traffic for various measures to improve overhearing (802.11g protected mode). Column 'no measures' describes default Softspeak. In column 'fixed=11b,' we select a fixed station to serve as destination for aggregated packets and let it associates using 802.11b. In column 'fixed=11b,optout,' we additionally let stations with overhearing problems opt out of downlink aggregation. The values given are the average and standard deviation MOS across all downlink VoIP sessions.

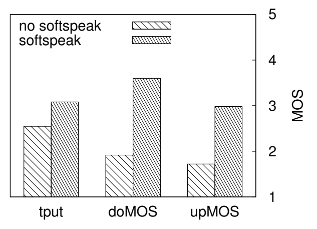
Softspeak enabled	No measures	Fixed=11b	Fixed=11b, optout
No	3.7 ± 0.095		
Yes	2.8 ± 1.0		
Yes, fixed Station 1	3.4 ± 0.63	3.5 ± 0.23	
Yes, fixed Station 2	2.7 ± 1.0	3.2 ± 0.81	3.5 ± 0.31

station as the destination for aggregated packets may greatly alleviate loss: picking a station that consistently experiences frame corruption causes the AP to often retransmit aggregated frames thereby increasing each station's probability of receiving a correct copy. For a particular choice of station (Station 1 in Table 2.3), we observe that the average downlink loss rate consistently reduces to below 2%, resulting in an average MOS of 3.4. However, the MOS variance remains high. Second, the selected station can be made to associate with the AP at a lower rate, causing aggregated packets to be transmitted at the lower rate and further reducing frame corruption. To test this, we force Station 1 to associate in 802.11b mode (*fixed=11b* in Table 2.3) and obtain a MOS of 3.5 as well as reduced variance. Note that to avoid condemning one of the stations to low-rate communication, a dummy 802.11 receiver can be added to the downlink aggregator box (or placed separately) and made to associate at the lower rate.

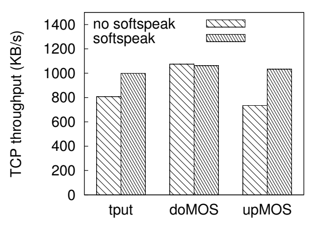
Our third measure is to have any remaining bad receivers opt out of the downlink portion of Softspeak. By de-registering with the aggregator, these clients receive separate VoIP frames as in the non-aggregated case (while continuing to measure loss rate from received aggregated packets to help decide whether and when to re-register). Note that these stations can still participate in uplink aggregation. To demonstrate that such a scheme can gracefully address this situation in practice, we evaluate all three measures when making a poor choice for the fixed station: Station 2 in Table 2.3, which gives a



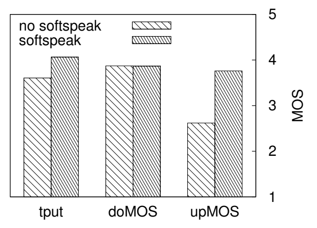
(a) TCP downlink, 10-ms codec



(b) TCP uplink, 10-ms codec



(c) TCP downlink, 20-ms codec

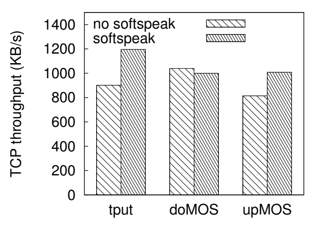


(d) TCP uplink, 20-ms codec

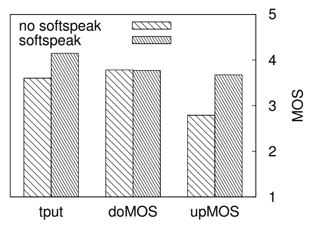
Figure 2.11: 10-ms and 20-ms VoIP codecs in combination with TCP traffic (802.11g protected mode, two stations opt out).

low MOS value of 2.7. After making the fixed station associate in 802.11b (improving average MOS to 3.2), we find that two stations consistently experience a high loss rate and low MOS. Once these two stations opt out of downlink aggregation, we arrive at a MOS of 3.5 with low variance (*fixed=11b,optout*).

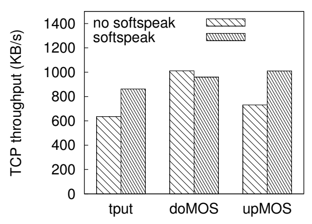
Of course, several of these measures have the potential of sacrificing much of the bulk traffic throughput gains that were obtained from downlink aggregation in the first place. We evaluate both TCP throughput and VoIP quality based on the above Station 2 and while applying all three measures. Downlink TCP throughput (Figure 2.11(a)) does not suffer much from these countermeasures: Softspeak continues to more than triple TCP downlink throughput. However, the resulting uplink TCP throughput (781KB/s, Figure 2.11(b)) is 12% less than the throughput achievable by Softspeak without enabling these countermeasures. Nevertheless, even with the countermeasures enabled Softspeak is able to achieve a significant improvement on residual throughput (34%) on TCP uplink traffic. For both TCP downlink and uplink Softspeak mostly maintains or significantly improves VoIP quality. For completeness, Figures 2.11(c) and (d) present the corresponding results when all clients use a 20-ms G.729 codec. As expected, Softspeak delivers less benefit in terms of throughput increase, yet remains critical for uplink VoIP call quality.



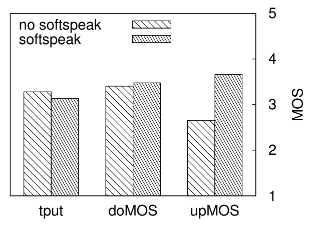
(a) TCP downlink.



(b) TCP uplink.



(c) TCP downlink+Web.



(d) TCP uplink+Web.

Figure 2.12: VoIP in combination with bulk TCP traffic (802.11g protected mode, no opt-out). Only five VoIP stations are active. In (c) and (d) the remaining five stations engage in Web traffic. The throughput measured is that of bulk TCP.

2.3.5 Softspeak and Web traffic

So far we have focused on Softspeak's impact on bulk traffic, without other traffic present. In reality, of course, one may expect a diverse traffic mix. We next evaluate how our results change in the presence of Web traffic, by running an equal number of VoIP clients and Web clients in combination with a bulk TCP stream, where each of the Web clients repeatedly downloads the front page of `cnn.com` (630 KB). Note that the size of our testbed limits us to five VoIP clients and five Web clients, and the magnitude of improvement is expected to be smaller than for a larger number of clients. In Figure 2.12, we plot Softspeak's improvements before (a and b) and after (c and d) adding Web traffic. Comparing the two scenarios we find that, independent of the presence of Web traffic, Softspeak (a) raises uplink MOS to an identical level, (b) roughly maintains downlink MOS, and (c) improves downlink TCP throughput to the same degree (roughly 35%). However, we also find that the gains made by Softspeak on TCP uplink throughput diminish in the presence of Web traffic. In summary, it appears that, with the exception of TCP uplink throughput, Softspeak's improvements on the efficiency of the network are maintained, even when Web traffic is present.

2.4 Limitations and discussion

The scalability of Softspeak is limited by the number of slots available for uplink TDMA, *i.e.*, ten clients in 802.11b (given 10-ms inter-packet interval VoIP codecs). In 802.11g (non-protected mode) the number of clients can be raised to twenty by choosing 500-μs TDMA slots (assuming a 48-Mbps sending rate). In addition, the number of available slots can be further doubled in the case that only 20-ms codecs are in use.

Softspeak relies on clients overhearing each other's VoIP communication to perform downlink aggregation. Therefore, if a WLAN uses a WiFi encryption protocol such as WPA2, downlink aggregation is no longer possible. Uplink TDMA, on the other hand, is not affected by encryption. Protocols encrypted above the MAC layer, such as Skype, can continue to take advantage of Softspeak's downlink aggregation, as long as they allow some way of being detected as VoIP.

Another consequence of downlink aggregation is that Softspeak places a station's interface in promiscuous mode, raising concerns of increased power usage. Stations engaging in VoIP traffic cannot currently benefit from 802.11 power saving mode (PSM) with or without Softspeak enabled, since PSM's duty cycling granularity is too coarse (a multiple of the beacon interval time). However, Softspeak introduces a well-defined schedule, both for uplink (TDMA) and downlink traffic (the aggregator's schedule), even in the face of jitter caused by VoIP applications or the wide-area network. Future rapid-duty cycling hardware may be able to exploit Softspeak to provide more fine-grained power savings.

VoIP silence suppression may go some way towards mitigating the impact of VoIP, decreasing the need for Softspeak. However, it appears that silence suppression is not universally implemented or supported by all codecs. For example, while monitoring a G.711 call between a Linksys VoIP phone and a softphone (Twinkle), we observe no change to inter-packet time in traffic sent by either side, even when the sender is muted. The same applies when we monitor a SkypeOut call. On the other hand, we have observed that Skype-to-Skype calls do employ silence suppression by lowering the sending rate, rather than eliminating traffic completely.

2.5 Softspeak as a black box approach

We now discuss the challenges we face by implementing Softspeak using a black box approach and the extent to which our Softspeak implementation satisfies the three criteria for an effective black box approach laid out in Chapter 1. Since uplink TDMA and downlink aggregation use very distinct mechanisms, we discuss them separately.

2.5.1 Uplink TDMA

In order to implement a TDMA schedule that incorporates a reasonable number of VoIP stations (*e.g.*, ten VoIP users sharing a hotspot), it is crucial to control packet transmission at a sufficiently small time scale (*e.g.*, 1 ms). Doing so using a black box approach, *i.e.*, making changes to neither the 802.11 protocol nor 802.11 hardware, is challenging for the following reasons. 802.11 (as deployed) does not provide a way to reserve a given TDMA slot for a VoIP station and prevent the slot from being overrun by a data packet. An apparently straightforward way in which a VoIP station might handle this problem is to sense the channel and decide whether to transmit a pending VoIP packet depending on whether there is enough remaining time in the station's TDMA slot. However, a feedback loop based on this approach would have to be extremely short: 10 μs in order to seize the channel at highest priority. (If the highest priority is not used, the VoIP station needs to be able to cancel a scheduled packet in the event that the channel is seized by another station.) Current commodity 802.11 hardware does not permit such a short sensing–transmission feedback loop to be implemented in software. Instead, Softspeak eliminates the need for a feedback loop altogether by transmitting packets and manipulating 802.11 hardware backoff timers in accordance with a schedule. We have shown that given an idealized implementation (one in which software manipulates hardware with the best possible timing precision), a software-implemented, collision-free TDMA schedule for ten VoIP stations is feasible.

While no standardized interface to 802.11 hardware exists that permits software to adjust backoff timing, in practice some variant of such an interface is widely supported by current hardware, since it is needed by 802.11 implementations to comply with the 802.11e standard. We verify that uplink TDMA is indeed *portable* in this sense by adapting our implementation to both Ralink and MadWifi hardware. Next, while it is inevitable that a software implementation manipulates 802.11 hardware at less than perfect timing precision, the results in Section 2.3.2 demonstrate that the hardware interface used by uplink TDMA provides *sufficient control* to achieve close to optimal performance results. Finally, we verify that uplink TDMA interoperates with other 802.11 hardware: the 802.11 standard provides sufficient leeway for senders to transmit a packet using SIFS or SIFS + cwslot backoff, such that compliant receivers are able to receive the packet, and other stations are able to correctly sense that the channel is occupied. Experimentally testing the ability of our APs (both Avaya AP-8 and Linksys WAP4400N) to receive packets transmitted by a single VoIP station, we do not find any appreciable difference between enabling or disabling uplink TDMA in the VoIP station. We conclude that uplink TDMA is *protocol-feasible*.

2.5.2 Downlink aggregation

Like uplink TDMA, downlink aggregation requires changes to neither access points nor the 802.11 hardware in 802.11 clients. In contrast with uplink TDMA, downlink aggregation does not control 802.11 hardware directly. Rather, its control interface is a combination of the IP protocol (sending and receiving encapsulated IP packets) and packet sniffing (overhearing), the former of which is universally available, the latter commonly available. In Section 2.3.4, we extend the control interface to include the ability for a VoIP station to associate at a lower MAC rate, again a feature commonly available in 802.11 hardware. We conclude that the interface is sufficiently *portable*. Next, while encapsulation by the aggregator modifies IP packets, it produces IP (and ultimately 802.11) packets that conform to the standard, and therefore downlink aggregation is *protocol-feasible*.

Finally, examining the extent to which downlink aggregation has *sufficient control* to achieve good performance, we observe a fundamental tradeoff between residual capacity and VoIP quality in the presence of imperfect overhearing: to maximize residual capacity, Softspeak needs to minimize the number of VoIP packet transmissions or maximize the MAC rate, which in turn may increase the error rate of VoIP traffic. We have explored several mechanisms in Section 2.3.4 that allow this tradeoff to be controlled, such as lowering the MAC rate. Previous work has proposed to use multicast instead of unicast [75, 76] — another means to decrease the error rate by using a lower MAC rate. However, multicast packets are sent at too coarse an interval to be usable for VoIP traffic. In other words, using multicast is not protocol-feasible for VoIP.

2.6 Related work

Researchers have studied VoIP call quality in wireless networks and attempted to quantify how many VoIP calls traditional WiFi networks can handle while maintaining various quality-of-service (QoS) metrics. These range from analytical and simulation-based studies [8, 50, 74, 81] to those that validate findings by measurements on actual experimental testbeds [10, 31, 63]. While precise findings vary, all studies agree that the effective VoIP capacity of a WiFi network is less than one might expect given the bandwidth usage of typical VoIP streams.

The poor performance of VoIP in WiFi networks is not protocol specific, but is symptomatic of a general issue with any CSMA (carrier-sense, multiple-access) network: channel access and arbitration becomes increasingly inefficient as load (in terms of number of attempted channel accesses) increases. TDMA can be far more efficient under heavy load. Indeed, 802.11 includes both a point coordination function (PCF) mode and a Hybrid Coordination Function Controlled Channel Access (HCCA), in which the AP explicitly arbitrates channel access. Unfortunately, very few deployed 802.11 networks employ these modes.

If one considers modifying the hardware, a variety of options exist. For example, researchers have proposed modifying 802.11 PCF [8, 42] as well as alternative ways of implementing 802.11e-like functionality [74]. Of course, non-backwards compatible modifications do not address the issue facing today's networks. Accordingly, researchers have proposed a variety of explicit time-slotting mechanisms, both within the context of infrastructure-based networks [35, 52, 54, 61, 65] and multi-hop mesh networks [46, 57].

MadMAC [61], ARGOS [46], and the Overlay MAC Layer (OML) [57] each propose to enable time-slotting on the order of 20 ms. Snow *et al.* [65] present a similar TDMA-based approach to power savings where each slot is of the order of 100 ms and requires changes at the access points themselves. These scheduling granularities are too coarse to effectively support most VoIP codecs. While software TDMA (STDMA) [35] proposes to do TDMA for all traffic, they focus particularly on the performance of VoIP. Their approach is a substantial and backward-incompatible modification to 802.11 that requires accurate clock synchronization. More significantly, each of the above schemes require the entire network to support the new TDMA architecture with no support for unmodified clients.

Over and above TDMA, the SoftMAC [52] and MultiMAC [26] projects also suggest modifications to 802.11 MAC behavior, including changing the ACK timing and modifying back-off parameters. The authors do not provide many details about their implementations, however, nor do they evaluate their scheme with deadline-driven VoIP traffic.

Focusing explicitly on improving the performance of VoIP traffic in mixed-use networks, various proposals have suggested prioritizing VoIP traffic [10, 81], notably a commercial product, Spectralink Voice Priority (SVP) [2]. SVP prioritizes downlink VoIP packets in the AP transmit queue and does not back-off when attempting VoIP transmissions. While we leverage similar optimizations, SVP does not do scheduling, thereby increasing collision rate due to the lack of back-off.

Others have proposed the concept of downlink aggregation in simulation [75, 76], encapsulating multiple VoIP packets into a single packet at a multiplexer node and using multicast to send the packet. Unfortunately, commodity access points typically transmit multicast frames at a multiple of the beacon interval, thereby introducing a delay that is unacceptable for VoIP traffic. The authors present analytical and simulation results for their scheme as well as their experience based on a three-node test-bed, however they do not implement or evaluate their scheme in a practical setting. In contrast, we present a detailed implementation and evaluation of a unicast based downlink aggregation scheme and furthermore demonstrate that downlink aggregation in itself does not fully remove excessive overhead of VoIP traffic in a mixed-use network.

Finally, several studies [44, 62] have shown using simulations that prioritizing traffic, using modified contention parameters, can lead to fairness and better resource allocation in both uplink and downlink directions. In contrast to our work, these proposals aim only to balance uplink and downlink traffic flows and do not evaluate TCP traffic in combination with VoIP traffic.

2.7 Acknowledgments

Chapter 2, in part, is a reprint of the material as it appears in Proceedings of the 6th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, April 2009. Verkaik, Patrick; Agarwal, Yuvraj; Gupta, Rajesh; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 3

FairEst

In the previous chapter, we used a cooperative model to improve performance of a mixed VoIP/data workload in 802.11: clients cooperate with each other and with wireless operators to simultaneously reduce the impact of VoIP on 802.11 and improve VoIP performance. In this chapter, we examine a scenario in which an 802.11 wireless operator is faced with a set of potentially selfish clients and wishes to establish fairness in its network. In such a scenario, the wireless operator cannot assume cooperation from clients, and we arrive at an adversarial model. However, enforcing fairness in a shared medium is fundamentally a hard problem — ultimately nothing can prevent a client from appropriating an arbitrary amount of airtime for itself. In this chapter, we instead address the more practical problem of enforcing fairness assuming the capabilities of a client running commodity hardware and existing wireless driver and TCP/IP implementations. We propose that a wireless AP uses a black box approach to control such wireless clients in order to enforce fairness.

In Section 3.1, we discuss basic fairness properties of 802.11 and the potential for clients running emerging upload-oriented applications to inadvertently or deliberately use an unfair share of airtime, demonstrating this point experimentally in Section 3.2. Also in Section 3.2, we examine the performance of a state of the art solution for enforcing fairness in 802.11, the Time-Based Regulator (TBR). While TBR uses a black box approach — controlling clients from the access point — we find TBR does not have sufficient control to solve the problem at hand, lacking an effective algorithm to estimate client demand. In Section 3.3, we propose FairEst, a black box approach to establishing fairness, based on a client demand estimating algorithm designed to have sufficient control over wireless clients. We describe and evaluate our prototype implementation of FairEst in Sections 3.4 and 3.5. We review related work in Section 3.6 and conclude this chapter by summarizing our black box criteria for FairEst in Section 3.7.

3.1 Motivation

In recent years, a number of bandwidth-hungry home networking applications have emerged, ranging from file sharing and back-up (*e.g.*, clients exchanging data with a Windows Home Server) to media streaming (*e.g.*, clients exchanging media with a TiVo or Slingbox). An important characteristic of these applications is that clients not only receive content, but also send content, a departure from the traditional assumption that clients mostly download content from the Internet. Furthermore, in addition to using wired Ethernet, clients of these applications may connect through wireless, typically through an 802.11 access point that is part of the home network. In fact, as mobile technologies such as laptops, netbooks, iPads, and WiFi enabled cell phones gain popularity, users are likely to increasingly avail themselves of the flexibility offered by wireless connectivity to access these applications. In the case of TiVo and Slingbox, even the media server may connect to the home network through wireless, eliminating the need to run a network cable to the media center. As a result, we expect that 802.11 home networks will increasingly be handling a workload in which wireless clients (including media servers) use substantial capacity to send data. More generally, as higher capacity wireless networking technology such as 802.11n becomes more widespread, users are increasingly likely to perform high bandwidth uploads through their wireless connection that otherwise would have been the preserve of wired networking: uploading media to the Web or by email, or even backing up content to the cloud [73].

Unfortunately, as is well known, 802.11's channel contention mechanism, the distributed coordination function (DCF), is unfair in the face of heavy uploaders, favoring uplink (from client to access point) over downlink communication. Essentially, DCF provides equal transmission opportunities to each station, regardless of whether the station is a client contending for itself or the access point contending to send downlink traffic to multiple stations. Since 802.11 uses a shared medium, clients that are mostly receiving traffic will receive an unfair share of the channel in the presence of a heavy uploader. Beyond DCF unfairness, we show that it is possible for a selfish uploading TCP user to grab a virtually unlimited amount of capacity from other users, by employing common unmodified TCP and 802.11 implementations. We note that the bottleneck that today constrains a home user's upload throughput, the provider's access link, does not exist for communication that remains within the home or is sourced at enterprise networks.

Thus, it appears that now more than ever there is a need to enforce fairness in 802.11 networks, *i.e.*, to ensure that clients receive equal access to channel resources, and to do so in the face of a set of autonomous, potentially adversarial clients. Following previous work, we argue for airtime fairness [67], as opposed to, *e.g.*, throughput fairness or fairness of access opportunity. Under airtime fairness, each client receives equal access to channel airtime, whether to send or receive packets. Previous work has shown that throughput fairness leads to reduced aggregate throughput when channel conditions vary across clients. The reason is that packets sent to or from a client experiencing poor channel conditions are more likely to be retransmitted or sent at a lower rate, thus consuming more airtime per byte transmitted. Ultimately, this discrepancy of airtime usage among clients causes aggregate performance of the channel to be dominated by poorly performing clients [38, 67]. In addition to the asymmetry between uploaders and downloaders discussed above, similar arguments can be made against fairness of access opportunity.

We approach the problem of fairness from the perspective of a wireless provider, who we assume has an interest in enforcing fairness in its network. While the clients of the provider's network are autonomous, the wireless provider at least has ownership of its access point. Thus, in this chapter, we apply a black box approach in which a provider's access point controls its clients as black boxes. Fairness in 802.11 has received a great deal of attention in the past, and many studies have proposed using access points to enforce fairness. However, to our knowledge no previous work exists that gives an access point *sufficient control* over its clients to establish fairness, given the upload-oriented applications we have described. Most such previous work has focused on access points enforcing fairness among *downlink* users. We speculate that the reason for this focus is that (a) achieving fairness among downlink flows was presumed adequate in the absence of heavy uploaders, (b) uploading clients are fundamentally harder to control: not only is it impossible for a DCF based access point to directly throttle uplink traffic, the access point does not even have the ability to observe the demand of a client (*e.g.*, its queue length).

Studies that do consider fairness in the presence of heavy uplink users are limited in one or more of the following ways. Some do not arbitrate among different uplink users, treating all uplink users as a single aggregate user. More importantly, the algorithms in this class only solve part of the problem of regulating uplink traffic: they are able to throttle (TCP) uplink traffic but lack a mechanism to estimate a client's demand. In particular, these algorithms cannot determine whether a client with low airtime utilization has low demand or has high demand but is sensitive to the presence of other traffic. Only in the latter case should the client be allocated more airtime.

By contrast, we present *FairEst*, a black box approach to fairness consisting of a fairness algorithm that arbitrates among a mix of uplink and downlink users and is robust against selfish behavior by uploaders. At the core of our algorithm is a novel client demand estimation technique that enables the algorithm to efficiently trade off fairness with utilization. Moreover, we show that *FairEst* can be implemented as part of an access point based on commodity hardware. Like many other approaches, we limit the scope of our work to uplink traffic that responds to feedback (*e.g.*, TCP) but demonstrate that it is effective in the presence of TCP flows based on an aggressive TCP congestion control implementation and a large maximum congestion window.

3.2 Experimental observation

In this section, we present the results of a number of experiments that demonstrate the impact an uploading station may have on a wireless network. We start with a basic experiment in which all wireless clients perform bulk TCP transfers. Next, we replace the bulk transfer workload with several other traffic types one may expect to find in a home or enterprise environment: Web, SMTP, and SMB traffic. We first show results for a simple default implementation of the AP, one based on a FIFO queue. We find that a selfish uploader is able to appropriate an arbitrary amount of channel capacity at the expense of other users. Then we re-run our experiments, enabling our implementation of the time-based regulator (TBR) [67] on the AP, a recent approach to enforce fairness in 802.11. While TBR can substantially improve fairness in some cases, we find that its performance is not robust when exposed to a variety of workloads.

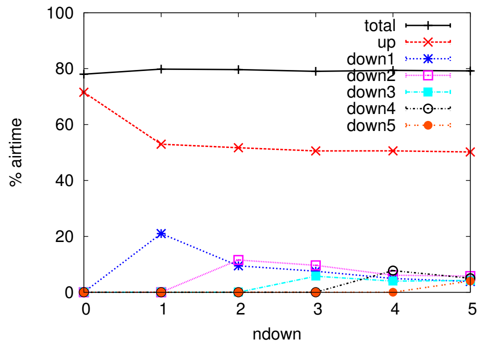
Our wireless testbed consists of six clients and one access point (AP), spread over two neighboring offices in a university building. One of the wireless clients is an uploader, the remainder are downloaders. In each configuration, we vary the number of downloaders, generating a separate 90-second experiment for each downloader count. Unless specified otherwise, in each experiment we give downloaders a ten-second head-start, before starting the uploader. We experiment with regular and aggressive uploaders by varying the uploader's 802.11 contention settings. Our aggressive uploader represents a selfish client that uses readily available command line tools to improve its performance at the expense of other clients. While we present results for a fixed uploader, we have verified that the results are practically identical when we change the location (in particular the room) or identity of the uploader.

In most of our experiments, our stations use TCP CUBIC [37] as the congestion control algorithm, an aggressive algorithm designed to recover quickly from losses in high latency networks. This is a natural choice for the uploader, given that we wish to explore the impact of an aggressive uploader based on commonly available implementations. By also letting downloaders use TCP CUBIC, we ensure that our results are not simply dependent on the choice of congestion control algorithm, but actually a function of the 802.11 contention settings. Similarly, we use large maximum TCP congestion window sizes, such that throughput is never congestion window limited. Further details of our experimental setup appear in Section 3.5.

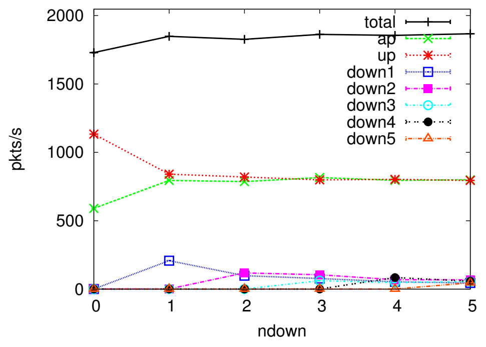
3.2.1 Bulk workload

Figure 3.1 shows per-client airtime usage and packet rate for the case of a bulk workload and the AP configured to use our implementation of a FIFO queuing discipline (as is the default in Linux). We have implemented our own version of FIFO so that we can instrument it with measurement and debugging output; the measured behaviour is essentially identical to Linux's default or indeed to that of an unmodified commercial AP (Linksys WRT54g). The queue length is set to 199, Linux's default for this interface.

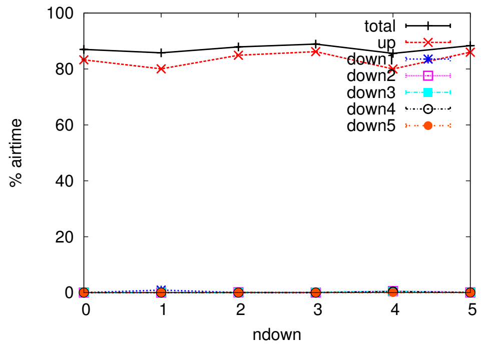
In Figure 3.1(a) all clients use the same (non-aggressive) 802.11 settings to contend for the channel as the access point. As expected, DCF favors the uploader: the uploader retains a close to constant share of airtime independent of the number of downloaders present. The reason for this behavior is apparent from Figure 3.1(b): independent of the number of downloaders, the uploader and the AP contend for the channel equally and send packets at equal rate. However, whereas the uploader only sends TCP data packets for itself, the AP sends a mix of TCP acknowledgments (ACK) packets to the uploader and TCP data packets to the downloaders. The downloading clients themselves



(a) Non-aggressive uploader (airtime)



(b) Non-aggressive uploader (packet rate)



(c) Aggressive uploader (airtime)

Figure 3.1: Performance of a varying number of bulk downloaders (*downx*) combined with one bulk uploader (*up*). The AP uses a FIFO queuing discipline. Figures (a) and (b) present the results of an experiment in which all stations contend normally. Figure (a) shows per-client airtime usage, *i.e.*, the sum of uplink and downlink airtime, while Figure (b) shows per station (client or AP) packet rate sent (*i.e.*, uplink and downlink are not combined). Figure (c) shows per-client airtime usage for an experiment in which the uploader contends aggressively.

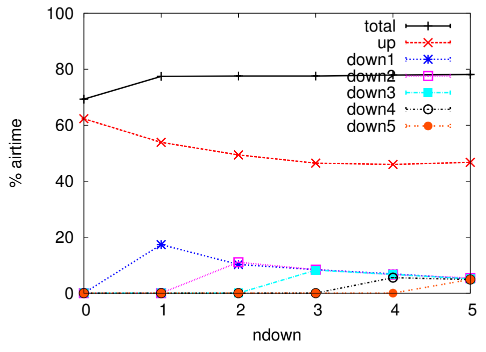
send TCP ACK packets at a much lower rate, since TCP transmits only one ACK per two data packets received. For the same reason, in the case that no downloaders are present (ndown=0), the uploader sends packets (data packets) at a higher rate than does the AP (ACK packets).

These results confirm that by default 802.11 favors an uploader. However, it turns out that an uploader can occupy effectively all channel capacity if it wishes to, as shown in Figure 3.1(c), where we change the 802.11 contention settings for the uploader to be aggressive. We note that a user can make such changes by means of readily available command line tools. We have verified in separate experiment that changing the contention settings of *all* wireless clients to be aggressive produces results identical to Figure 3.1(c), suggesting that there is little a downloading wireless client can do to protect itself from an aggressive uploader.

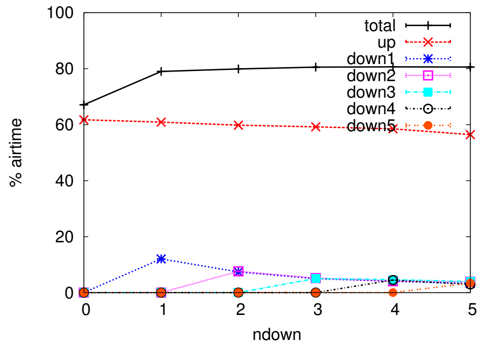
3.2.2 Other workloads

To investigate the dependence of fairness on the type of workload in the wireless network, we present experiments pitting SMTP or SMB (uploader) against Web traffic (downloaders). In the SMTP experiments, we select one of the wired hosts to be both an SMTP server (postfix 2.5.1) and a Web server (apache 2.2.8) and verify that during our experiments this host is never overloaded. The workload of the uploader is to send a large (8 MB) email to a gmail account, relayed through the SMTP server. The workload of each Web client is to download a copy of the front page of `cnn.com` hosted at the local Web server, a collection of 81 objects, totaling 650 KB. We use `wget` and enable persistent connections. Figure 3.2(a) shows that the SMTP client by itself uses about 60% of airtime, compared to close to 80% for a bulk uploader (Figures 3.1(a) and (c)). We attribute this to the large overhead of DNS and SMTP negotiation, both of which are limited by round-trip time. Our Web traffic is further limited by round-trip time, since many of the Web objects are small (over 85% are smaller than 10 KB). Hence, as seen in Figure 3.2(c), a Web client by itself does not occupy more than 40% of airtime. When combining both SMTP and Web traffic, the uploader retains most of its maximum airtime, while the downloaders lose most of theirs. In particular, a selfish uploader retains nearly all of its airtime (Figure 3.2(b)).

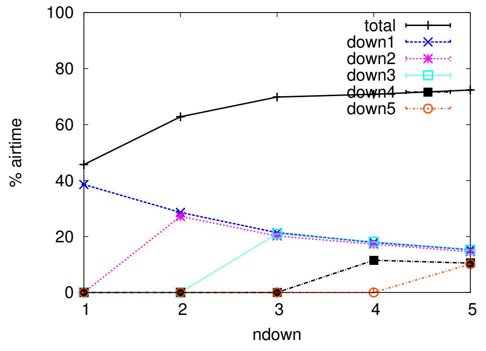
To measure the impact of an SMB uploader, we replace the uploader by a Windows Vista PC and connect it to a wired PC running Windows Home Server. The Vista



(a) Non-aggressive SMTP client

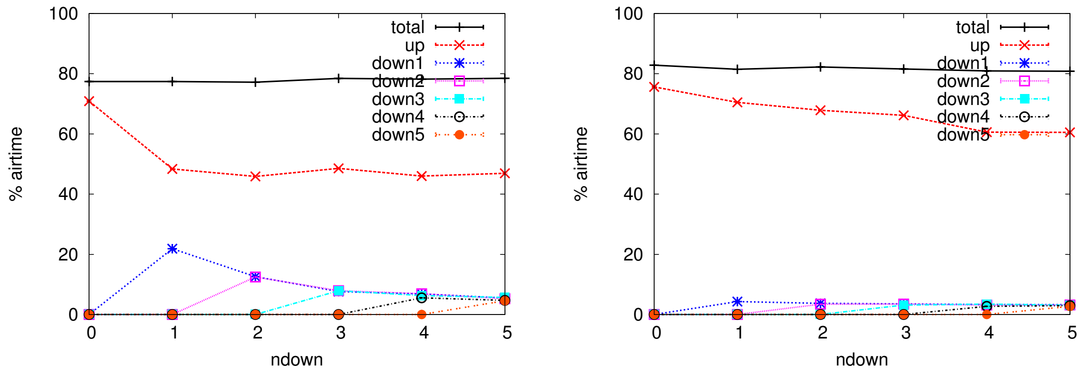


(b) Aggressive SMTP client



(c) Without SMTP client

Figure 3.2: An SMTP client (*up*), competes with a varying number of Web clients (*downx*). The AP uses a FIFO queuing discipline. The figures show per-client airtime usage.



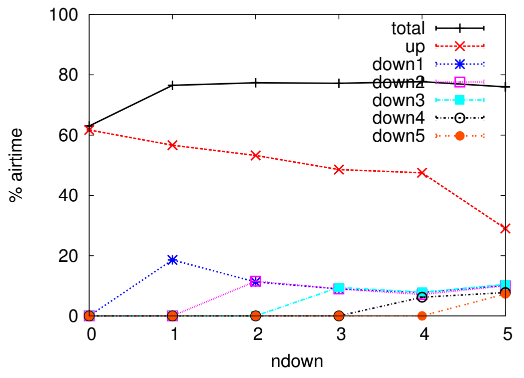
(a) Non-aggressive SMB client (b) Aggressive SMB client

Figure 3.3: An SMB client running Windows Vista (*up*), competes with a varying number of Web clients (*downx*). The AP uses a FIFO queuing discipline. The figures show per-client airtime usage.

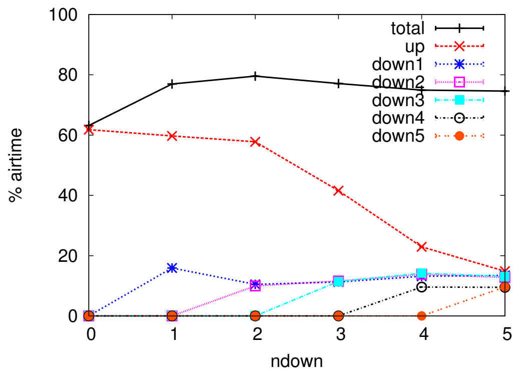
PC's traffic is routed through an Alix device configured with the desired aggressive or non-aggressive contention settings. We run `speedguide`, a popular TCP optimizer tool, on the Home Server, adopting the settings recommended by the tool. On the Vista PC, we enable Compound TCP (a Windows alternative to CUBIC [68]). The upload workload in this experiment is to copy ten MP3 audio files (total 57 MB) from the Vista PC to the Home Server. Figure 3.3 again confirms that aggressive settings allow an SMB uploader to dominate airtime usage.

3.2.3 Time-based regulator

The Time-Based Regulator (TBR) [67] is a recently proposed fairness algorithm that arbitrates among uploaders and downloaders and is implemented on the AP. (In the case of cooperative clients, TBR may additionally be implemented on wireless clients, however, since our focus is on unmodified, adversarial clients, we limit our interest to the AP-only version of TBR.) In TBR, the AP maintains a packet queue and a token bucket for each client, each token representing one μs of airtime. For each packet sent to or received from a client, the AP computes the airtime used by the packet and removes that from the client's token bucket. When sending packets, the AP only considers clients that have a positive token count. The fill rate of a token bucket therefore determines a client's airtime rate limit. Initially, each client's token bucket is filled at a rate corresponding to $\frac{1}{n}$ share of airtime (where n is the number of clients), or $\frac{1}{n}$ million tokens per second.



(a) Non-aggressive SMTP client



(b) Aggressive SMTP client

Figure 3.4: An SMTP client (*up*), competes with a varying number of Web clients (*downx*). The AP uses TBR as its queuing discipline. The figures show per-client airtime usage.

To improve work conservation, the AP periodically adjusts token bucket fill rates by transferring rate from clients that on average under-utilize their allocated rate to other clients. Note that while the AP cannot directly control packet transmission by uploading clients, it implicitly limits TCP uploaders by regulating TCP ACKs.

The authors of TBR do not provide specific values for some of TBR's parameters, thus we experiment with various parameterizations, in particular the token bucket size (100 K tokens and 25 K tokens), as well as the interval over which rates are measured and rate adjustment takes place (50 ms, 125 ms, 250 ms, 500 ms, 750 ms and 1 s). In Figure 3.4, we show the results of the parameterization that produces the best fairness solution for the SMTP/Web traffic experiment at ndown=5: 100-K token buckets and a 50-ms rate-adjustment interval. Comparing with FIFO in Figure 3.2, we see that TBR has little impact for lower values of ndown. Inspecting the results of this and other parameterizations more closely, we find that Web traffic does a worse job of using its allocated rate than SMTP traffic. Thus, Web traffic is frequently penalized by TBR for underutilization, despite the fact that it is operating below its demand. The reason that TBR nevertheless produces a fair solution in Figure 3.4(b) for ndown=5 is an artifact of the small rate adjustment interval. Rates measured at 50-ms time intervals have large variance when many clients are active, causing each client, including the SMTP client, to frequently fall below its allocated rate. Ultimately, the SMTP client still gathers enough allocated rate to produce an unfair solution, but takes increasingly longer to converge as

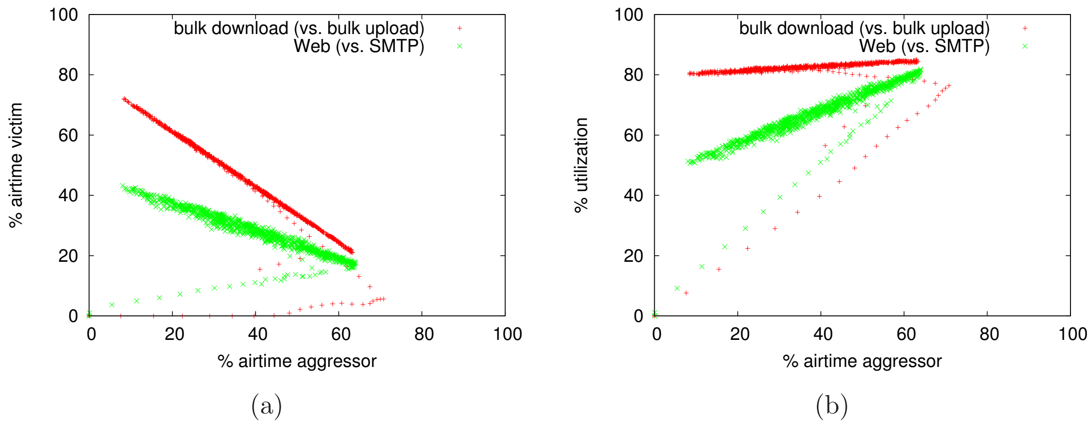


Figure 3.5: Airtime usage of a *victim* client as a function of airtime usage of an *aggressor* client for two workload configurations (scatter plots). In red, the victim and aggressor are a bulk downloader and uploader, respectively. In green, the victim runs Web traffic and the aggressor SMTP traffic. Figure (a) shows airtime of the victim vs. airtime of the aggressor. Figure (b) shows channel utilization vs. airtime of the aggressor. Samples are taken once every 500 ms and further averaged using an EWMA (exponentially weighted moving average) that gives each a new sample a 10% weight. For both workloads, but especially for the Web victim, the victim uses less than its maximum demand worth of airtime, even if the additional airtime needed is available.

the number of Web contenders increases, leading to lower average airtime results for *up* in Figure 3.4. Similarly, while an SMTP client ultimately appropriates more airtime as it becomes more aggressive, convergence time also increases, again reducing the average airtime results for *up* in Figure 3.4(b).

In order to sufficiently control a set of clients so as to achieve fairness, an access point needs not only a mechanism to throttle the sending rate of a client, but also a means to estimate its demand. We claim that the fundamental reason TBR does not achieve fairness in most cases is that it lacks an effective mechanism to estimate a client's demand. To effectively trade off utilization and fairness, a fairness algorithm faced with a low airtime using client must determine whether the client's low airtime usage is due to the presence of other traffic (lowering available capacity and/or increasing RTT), or whether the client is limited by its own demand. In the former case, airtime fairness can be improved by allocating more airtime to the client. In the latter case, doing so merely takes away airtime from other users without improving fairness, leading to unnecessarily low channel utilization. When TBR observes that a client is under-utilizing its allocated airtime share, it effectively assumes that the client is operating at its maximum demand. However, in practice, a client may be limited by factors other rather than available airtime.

To demonstrate this point, we run an experiment in which we measure the amount of airtime used by a *victim* client in response to airtime usage by an *aggressor* client. We configure the victim and aggressor with default and aggressive contention settings, respectively, and vary the amount of airtime used by the aggressor by setting different rate limits on the aggressor, each held for ten seconds at a time. Figure 3.5 shows this experiment for two different workloads. (Note that although we fix each aggressor rate limit for a duration of ten seconds, TCP's fluctuations produce many intermediate points.) The figure shows that the victim uses less than its maximum demand of airtime, even if the airtime needed is available to it. For example, when the SMTP aggressor is rate limited and uses 40% airtime, the Web victim is able to use only 30% airtime. The 10% airtime that the Web client would need to meet its demand of 40% is available but not used.

3.3 FairEst: a demand estimating fairness algorithm

In this section, we describe our black box solution to AP enforced fairness, which we call *FairEst*. Like TBR, FairEst controls client airtime usage based on a per-client rate limit (a token bucket). However, unlike TBR, FairEst includes an algorithm that estimates client demand, allowing it to adjust the per-client rate limit to achieve fairness without unduly sacrificing utilization.

FairEst is implemented in the software of an AP and treats the AP's clients as black boxes. The algorithm makes the following assumptions about clients. First, similar to TBR, FairEst assumes that a client's total airtime usage is dependent on the amount of downlink airtime the client receives from the AP, allowing the AP to control the airtime usage of TCP uploaders, as well as TCP or UDP downloaders. Second, FairEst makes several assumptions about client airtime usage (*e.g.*, being monotonically decreasing in the total amount of airtime used by all other clients), allowing FairEst to iteratively adjust airtime allocations, correcting previous mistakes.

```
 1. $S$ = { all clients }
 2. $RateLimits$[] = empty map
 3. $DemandEstimates$[] = empty map
 4. while(forever) {
 5.   sleep($interval$)
 6.   $U$ = 0; $Free$ = $\emptyset$
 7.   for client $s$ in $S$ {
 8.     compute $A_s$
 9.     $U$ += $A_s$
10.     if ($s$ did not reach a rate limit)
11.       $Free$ = $Free \cup \{s\}$
12.   }
13.   for client $s$ in $Free$
14.     $DemandEstimates$[$s$] = EstimateDemand($A_s$, $U$)
15.   ComputeRateLimit($RateLimits$, $DemandEstimates$, $S$)
16. }
```

Figure 3.6: The FairEst algorithm.

3.3.1 Basic FairEst algorithm

At a high level, FairEst follows the algorithm shown in Figure 3.6. (The EstimateDemand and ComputeRateLimit functions are discussed below.) Every *interval* ms, FairEst computes the (combined uplink and downlink) airtime usage A_s for each client s since the previous iteration, as well as the total channel utilization U (Lines 8 and 9). Based on these measurements, FairEst updates its estimates of each client's demand, using a model of a client's airtime function (EstimateDemand function, Line 14). Using the demand estimates, and again based on the model, the algorithm then computes a rate limit on clients' airtime usage in the ComputeRateLimit function (Line 15). By rate limiting a client, FairEst loses the ability to update the client's demand estimate. Therefore, ComputeRateLimit only rate limits a subset of clients. In addition, in every iteration of Figure 3.6, FairEst only updates the estimated demands for those clients that did not hit their rate limit in the previous iteration (Lines 10–13). ComputeRateLimit derives a rate limit in such a way that fairness will result under the assumption that the

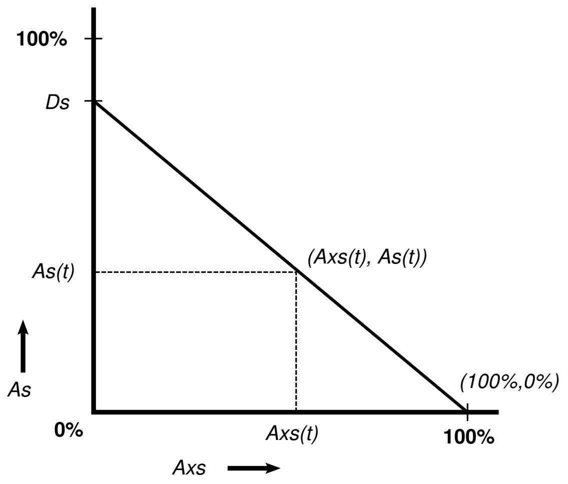


Figure 3.7: Model used to estimate a client's demand: a station s's airtime usage A_s is a linear function of $A_{\neq s}$, the sum of airtime usage by all other clients. It includes the points (100%,0%) and (0%, D_s), where D_s is the demand of s. By measuring A_s and $A_{\neq s}$ at time t, FairEst can estimate D_s.

models used by FairEst are correct. In practice, the models are only approximations. However, later in this section we show that under several reasonable assumptions FairEst is able to adjust for previous mistakes.

Next, we discuss the model FairEst uses to estimate demand, the implementation of the EstimateDemand and ComputeRateLimit functions, and FairEst's self-correcting behavior.

EstimateDemand function

To estimate demand, FairEst approximates a client's airtime usage A_s as a linear function of the total airtime usage of all other clients $A_{\neq s}$ (Figure 3.7). The model assumes the client's airtime usage decreases linearly from its demand D_s (when $A_{\neq s} = 0$) to 0 (when $A_{\neq s} = 100\%$), or $A_s = (1 - A_{\neq s}) \cdot D_s$. We can estimate client s's demand at time t by measuring $A_s(t)$ and $A_{\neq s}(t) = U(t) - A_s(t)$, where $U(t)$ is the total channel utilization. The estimate $E_s(t)$ for the client's demand is then the intercept with the A_s axis of the line through $(100\%, 0)$ and $(A_{\neq s}(t), A_s(t))$. In summary, the EstimateDemand function computes $E_s(t) = A_s(t)/(1 - A_{\neq s}(t)) = A_s(t)/(1 - (U(t) - A_s(t)))$.

From Figure 3.5 it is apparent that, at least on average, a linear relationship between A_s and $A_{\neq s}$ is a reasonable assumption for the workloads shown. Of course, we

```
ComputeRateLimit($RateLimits$, $DemandEstimates$, $S$)
{
  $s_{min}$ = $argmin_{s \in S}$ $DemandEstimates$[$s$]
  $r$ = 1 / ( 1/$DemandEstimates$[$s_{min}$] + (|$S$|-1) )
  for client $s$ in $S$
    $RateLimits$[s] = $r$
  delete $RateLimits$[$s_{min}$]
}
```

Figure 3.8: The ComputeRateLimit function.

have no reason to expect that $(100\%, 0)$ is a valid point in this relationship (even though the Web vs. SMTP workload comes close), therefore it is likely $E_s(t)$ does not equal the client's actual demand D_s. However, as we show later, $E_s(t)$ serves as an adequate approximation.

ComputeRateLimit function

In order to achieve airtime fairness, we wish all clients to use the same amount airtime. In principle, there is a global rate limit r that we can impose on the clients to achieve airtime fairness. We are interested in finding the highest such rate limit that still achieves fairness: setting r too high does not achieve fairness (unless all clients have similar workload); setting r too low achieves fairness, but reduces utilization unnecessarily. In fact, it is sufficient to rate limit all clients but one to r, as will become apparent below.

Figure 3.8 shows the implementation of ComputeRateLimit. To derive r, we use the same linear model as above. We compute r such that if we rate limit every client but the lowest demand client s_{min} to r, the linear airtime function of s_{min} dictates that $A_{s_{min}}(t+1)$ also equals r. In other words, we set $A_s = r$, $A_{\neq s} = (|S| - 1) \cdot r$ and $D_{s_{min}} = E_{s_{min}}(t)$ and derive $r = 1/(1/E_{s_{min}}(t) + (|S| - 1))$. It can be shown that setting r any higher causes s_{min} to receive less airtime than other clients. Therefore, r is the rate limit that achieves fairness while maximizing utilization and is the value computed and returned by the ComputeRateLimit function.

ComputeRateLimit depends on the estimates of client demands to compute r.

However, once ComputeRateLimit rate limits a client, FairEst can no longer make updated estimates of the client's demand — any measurement of A_s is simply the result of the rate limit. (In Figure 3.6 this is implemented by only updating estimates for clients in $Free$.) Fortunately, this is not a problem. First, ComputeRateLimit does not rate limit s_{min}, and therefore FairEst updates the demand estimate for s_{min} in its next iteration. Second, if a rate limited client s lowers its airtime usage and is no longer limited by r, it becomes a member of $Free$, and FairEst resumes updating its demand estimate. Finally, consider a rate limited client s that remains limited by r (is not a member of $Free$). If, without the rate limit in place, FairEst would have computed a higher demand estimate for s than for s_{min}, FairEst would still base its computation of r on s_{min}, and the result is the same. If, on the other hand, FairEst would have computed a lower demand estimate for s than for s_{min}, then we know that it would have done so on the basis of $A_s < A_{s_{min}}$. However, in this case, s could not have been limited by r.

Self-correction

If client airtime functions corresponded exactly to Figure 3.7, FairEst would make correct estimates and arrive at the correct solution (fairness with maximum utilization) in a single iteration. However, in using a black approach, it is impossible for FairEst to know the precise form of a client's airtime function. Thus, in practice, a client's airtime function is likely to depart from the model, in which case a single iteration of FairEst will fail to compute a correct solution. Fortunately, it is possible to show that FairEst will self-correct during the time that the client airtime functions remain stable, under two assumptions about client airtime usage: (1) a client's airtime function is monotonically decreasing; (2) for any pair of clients, one client's airtime is consistently larger than the other's.

Let a be the client whose airtime function is dominated by all others (in the sense of Assumption (2)). In Appendix B, we prove the following. First, FairEst is able to detect that a has the lowest demand. Subsequently, while FairEst is likely to overestimate or underestimate a's airtime usage A_a, it corrects for that in a subsequent iteration. Specifically, if in some iteration FairEst overestimates (underestimates) A_a, causing A_a to drop below (rise above) other clients' airtime usage, then in the next iteration, FairEst either shrinks the maximum difference between a and other stations (the difference between r and A_a), or else FairEst starts underestimating (overestimating)

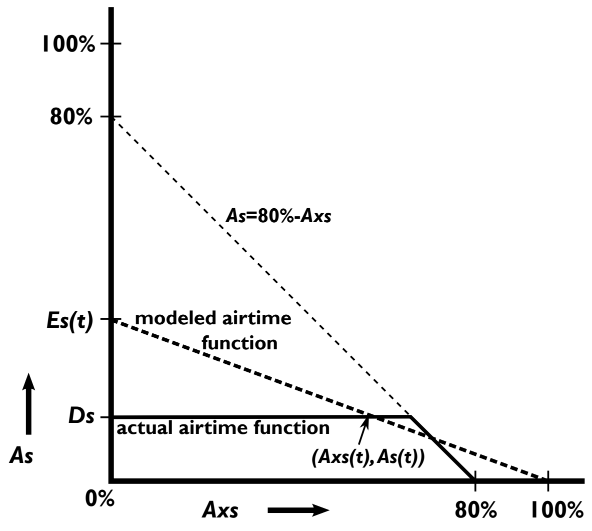


Figure 3.9: FairEst over-estimates an insensitive client s's demand. In the actual airtime function of s, A_s is insensitive to $A_{\neq s}$, until the point that 80% capacity is used, approximately the maximum channel capacity. When FairEst measures point $(A_s(t), A_{\neq s}(t))$, its estimate of client demand $E_s(t)$ may vastly exceed actual demand D_s.

A_a, giving A_a the largest (smallest) amount of airtime. While this behavior does not guarantee that airtime usage (averaged over multiple iterations) is fair, our evaluations in Section 3.5 show that, in practice, FairEst comes very close.

3.3.2 Balancing fairness and utilization

We next consider the impact of clients whose airtime usage is small, in particular, clients that are *insensitive* to the airtime usage of other clients. An example of such a workload is VoIP traffic, typically consisting of low volume UDP flows. While an insensitive client has a linear airtime function, a fit through $(100\%, 0)$ is very poor (Figure 3.9), and, for small flows, FairEst is likely to vastly overestimate the demand of the client before converging. Insensitive clients create two further problems. First, an insensitive client with low demand (such as a VoIP call), causes FairEst to drive down channel utilization (during convergence) as it rate limits other clients to the insensitive client's rate. Second, an insensitive client does not even benefit from FairEst rate limiting other clients, therefore any such rate limiting needlessly lowers channel utilization.

Independent of whether a client is insensitive, a low airtime client may cause

FairEst to strongly decrease channel utilization, perhaps more than a network operator is willing to sacrifice for the sake of fairness. We now describe a variant of ComputeRateLimit that permits a wireless operator to specify to FairEst a minimum utilization (MIN_UTIL) for the network. While this variant handles the case of low airtime, insensitive clients, it still does so by sacrificing some network utilization (by targeting a utilization of MIN_UTIL) and does not handle the case of high airtime, insensitive clients. We leave a more efficient solution, one that explicitly detects insensitive clients and reserves the right amount of airtime for them, as future work.

The variant of ComputeRateLimit that handles low airtime clients, shown in Figure 3.10, estimates the channel utilization for a computed rate limit, and, if the estimated utilization falls below MIN_UTIL (Line 5), proceeds to compute a different (higher) rate limit (Lines 6–19), one that will lead to a channel utilization of exactly MIN_UTIL. While for the rate limit thus computed, low demand stations will not receive as much airtime as other stations, low demand stations still receive more airtime than without any fairness solution in place.

Specifically, ComputeRateLimit calculates r, $reserved$, and a subset S' of S, such that S' contains the set of all clients limited by r, $reserved$ is the sum of airtime used by remaining clients, and utilization is MIN_UTIL. The utilization for such a set is $|S'| \cdot r + reserved = MIN\_UTIL$, where $reserved = \sum_{s \in S - S'} A_{s,mu}$, $A_{s,mu}$ being the predicted airtime usage of s when utilization is MIN_UTIL. For each client s that is not rate limited, the linear model predicts $A_{s,mu} = D_s \frac{1-MIN\_UTIL}{1-D_s}$, as computed in Lines 11–12. ComputeRateLimit checks that all members in its currently computed S' are indeed rate limited (Line 13), and if not, adjusts its current r, $reserved$ and S' variables to exclude such stations from S'.

3.4 Implementation

We implement FairEst as software modifications to a Linux 2.6.24-based access point (a PC in our case). The bulk of our implementation is contained within a Linux queuing discipline (*qdisc*) module. To control clients, we have implemented three components. First, we make minimal modifications to the access point's WiFi driver (MadWifi 0.9.4), so that it reports each packet transmission to the qdisc module: both the amount of airtime used by a packet and the client that sends or receives it. Second, we implement token buckets, similar to TBR [67], allowing us to rate limit clients. Third, we

```
ComputeRateLimit($RateLimits$, $DemandEstimates$, $S$)
{
  $s_{min}$ = $argmin_{s \in S}$ $DemandEstimates$[$s$]
  $r$ = 1 / ( 1/$DemandEstimates$[$s_{min}$] + ($|S|$-1) )
  if ($r \cdot |S|$ < $MIN\_UTIL$) {
    $S'$ = $S$; $reserved$ = 0; $done$ = 0; $r$ = 0; $s_{min}$ = 0
    while (! $done$ and $S'$ != $\emptyset$) {
      $done$ = 1
      $r$ = ($MIN\_UTIL$ - $reserved$) / $|S'|$
      for client $s$ in $S'$ {
        $airt$ = $DemandEstimates$[$s$]$\cdot$
                  (1-$MIN\_UTIL$)/(1-$DemandEstimates$[$s$])
        if ($airt$ < $r$) {
          $S'$ -= $\{s\}$
          $reserved$ += $A_s$
          $done$ = 0
        }
      }
    }
  }
  for client $s$ in $S$
    $RateLimits$[s] = $r$
  if ($s_{min}$)
    delete $RateLimits$[$s_{min}$]
}
```

Figure 3.10: Variant of ComputeRateLimit that handles low airtime clients.

implement the algorithm described in Section 3.3 to estimate client demands and based on that determine an effective rate limit.

Token buckets

To control the airtime usage of of clients, we use a similar implementation of airtime based token buckets as TBR [67], with one improvement: in addition to preventing a downlink transmission when a client's token bucket is empty, we also drop received uplink packets from such a client, finding that aggressive uploaders respond better to rate limiting in this manner. (We note that porting this improvement to TBR has little impact on the results for TBR in Section 3.2.) We choose a token bucket size of 100 K tokens. Token buckets are topped up at most once every 1 ms (triggered by packet transmission).

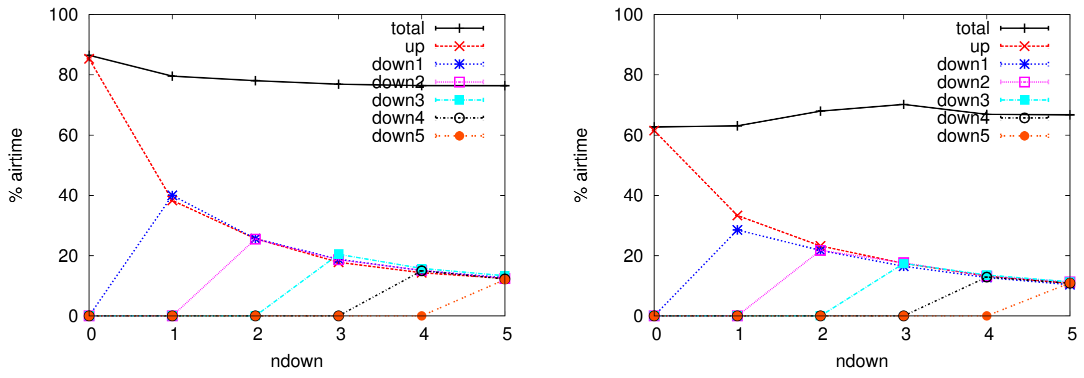
Estimating client demand

We implement the algorithm described in Section 3.3 to estimate client demands, using the following parameters. We run the algorithm every 500 ms (*interval* in Figure 3.6) — an interval sufficiently large to maintain low CPU utilization, yet small enough to let workload changes converge in seconds. In order to estimate a client s's demand (EstimateDemand function), we measure A_s and $A_{\neq s}$ each time the algorithm executes, *i.e.*, every 500 ms. To smooth fluctuations in estimated demand, we average the demand estimates using an EWMA, giving 65% weight to new samples.

3.5 Evaluation

In this section, we evaluate the degree to which FairEst is able to control wireless clients in order to achieve fairness. We also briefly evaluate the cost of using FairEst's black box techniques to derive a rate limit, vs an idealized approach in which FairEst knows the correct rate limit in advance. Finally, we evaluate FairEst's handling of low airtime clients.

Our wireless testbed consists of six clients and one access point (AP) spread over two neighboring offices in a university building. The wireless network runs in 802.11g mode set to a fixed MAC rate of 24 Mbps in order to ensure reproducibility of our results. Our set of wireless clients comprises four Alix 3d2 and two Soekris net4801 devices, running Linux 2.6.23 and 2.6.21.5, respectively. Our access point is an Intel Pentium 4 (1.80GHz) based PC that runs Linux 2.6.24.6. All wireless clients and AP communicate using Atheros AR5212 based 802.11 cards through a MadWifi 0.9.4 driver. In our ex-



(a) Bulk traffic (b) Web/SMTP traffic

Figure 3.11: An aggressive uploader, *up*, competes with a varying number of downloaders (*downx*). In Figure (a), the workload is bulk TCP for all clients. In Figure (b), the workload of the aggressive client is SMTP, while other clients run Web traffic. The AP uses FairEst as its queuing discipline. The figures show per-client airtime usage.

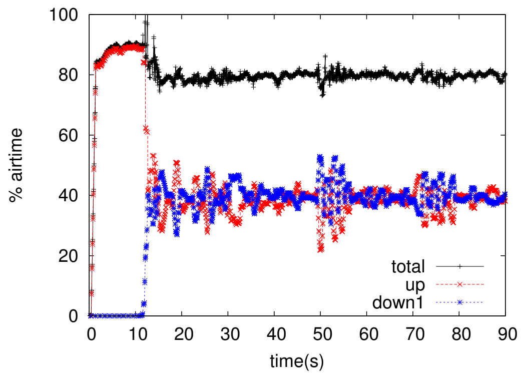
periments the wireless clients talk to a number of PCs (running Linux versions 2.6.16 or higher) and Soekris devices connected to the department's wired network through the AP. We measure airtime usage using Jigsaw [21] and have verified that fairness of TCP goodput (as measured at the AP) closely follows that of measured airtime.

We vary the aggressiveness of the uploader by changing the 802.11 contention settings of the client. Specifically, we modify the random backoff time that 802.11 uses as part of DCF collision avoidance (*cf.* Section 2.1.1) from the Atheros default of $9 \cdot (3 + rand(0, 15))\mu s$ to $9 \cdot rand(0, 3)\mu s$.

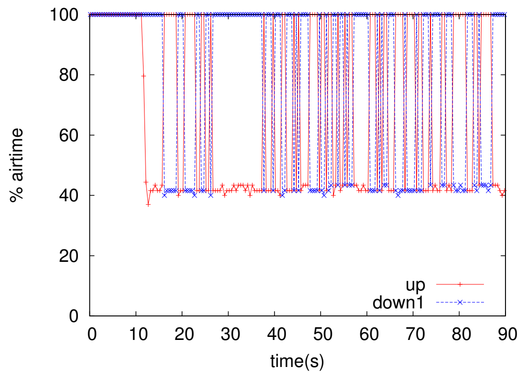
Our stations use TCP CUBIC [37] as the congestion control algorithm, unless specified otherwise. The Linux version of two of our wired PCs do not support CUBIC. Instead these PCs use the more aggressive BIC algorithm [79], so that the downloaders' congestion control algorithms remain at least as aggressive as that of the uploader.

3.5.1 Bulk and Web/SMTP traffic

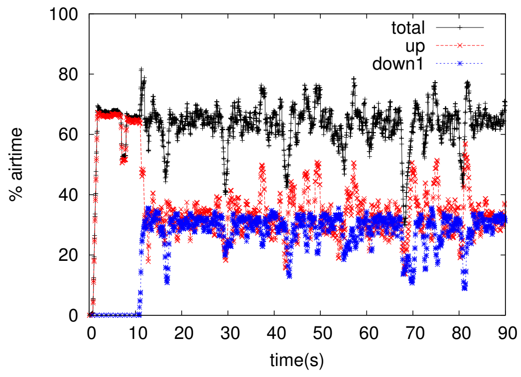
We begin by examining the performance of FairEst for bulk TCP and Web/SMTP traffic in Figure 3.11. Comparing Figures 3.11(a) and 3.1(c), FairEst comes very close to providing a fair solution for bulk traffic without sacrificing a large amount of channel utilization. Similarly, comparing Figures 3.11(b) and 3.2(b), FairEst's performance again approaches fairness for Web/SMTP traffic, but in this case sacrifices up to 20% channel



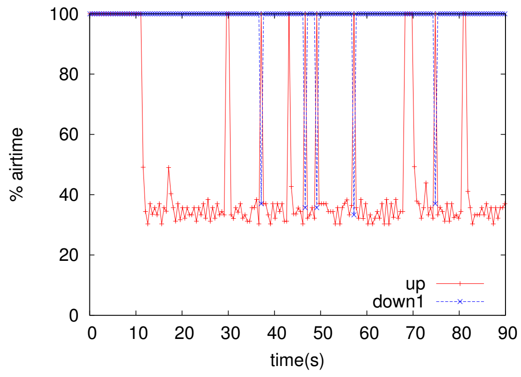
(a) Bulk traffic airtime



(b) Bulk traffic allocations



(c) Web/SMTP traffic airtime



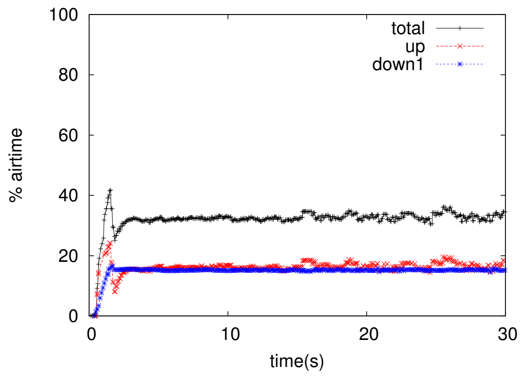
(d) Web/SMTP traffic allocations

Figure 3.12: Time series of airtime usage and FairEst's rate limits for ndown=1.

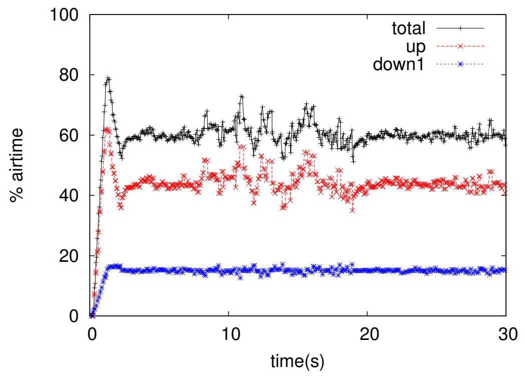
utilization. Note that for both workloads, for ndown=1, the airtime distribution and channel utilization correspond to what we expect based on the experiment shown in Figure 3.5.

Figures 3.12(a) and (c) show a time series of airtime usage for ndown=1 and Figures 3.12(b) and (d) the respective rate limits set by FairEst. After the first ten seconds (during which the aggressive uploader runs by itself), FairEst oscillates around an approximately fair solution, alternating its choice of which client to rate limit, and averaging to the results (for ndown=1) in Figure 3.11. FairEst reaches this state in a matter of seconds.

FairEst's oscillating behavior is a direct result of our use of a black box approach. Since FairEst lacks perfect knowledge of a client's airtime function, it is expected to repeatedly underestimate or overestimate a client's airtime usage. However, after creating an incorrect estimate, FairEst is able to correct itself and compensate for its mistake.



(a) Min. util. protection disabled.



(b) Min. util. protection enabled.

Figure 3.13: FairEst's effectiveness at handling of low airtime clients. *down1* receives a CBR (constant bitrate) flow of 3 Mbps. *up* performs a bulk upload and uses aggressive contention settings. In Figure (a), we have disabled FairEst's protection of minimum utilization, while in Figure (b), it is enabled.

It might be expected that such oscillating behavior could cause inefficiency. However, as we have noted above, the airtime distribution and channel utilization for ndown=1 correspond to the results of the controlled experiment shown in Figure 3.5, suggesting that any overhead caused by FairEst oscillating behavior is small.

3.5.2 Low airtime clients

Finally, we examine FairEst's effectiveness at handling clients with low airtime usage. Without special handling, we expect FairEst to reduce channel utilization to an arbitrarily low level in order to achieve fairness. However, using our modifications in Section 3.3.2, FairEst should attempt to maintain a minimum channel utilization of MIN_UTIL. Figure 3.13 confirms this expected behavior. Without protecting minimum utilization, FairEst rate limits *up* to the same airtime usage as *down1*, a low airtime user. As a result, utilization drops below 40%. When we enable protection of minimum utilization, FairEst predicts that striving towards fairness would lower channel utilization to below the MIN_UTIL threshold (60% airtime) and instead rate limits *up* only to the extent needed to reach a utilization of 60%. Figure 3.13 shows that FairEst succeeds at maintaining a channel utilization of approximately 60%.

3.6 Related work

Fairness in 802.11 has been studied extensively, with particular attention paid to the 'performance anomaly' in a multi-rate 802.11 network [38, 67], which causes all hosts to experience a throughput similar to that of the lowest MAC rate host, and unfairness between uplink and downlink flows [14, 36, 45, 56, 78].

To mitigate against unfairness, several studies follow a non black box approach, assuming cooperative clients or modifications to 802.11. Vaidya *et al.* apply the concept of fair scheduling to wireless LANs, letting clients adapt their 802.11 backoff settings to achieve fairness [71]. Similarly, Tan *et al.* propose to modify 802.11, so that clients use a larger *cwmin* contention parameter if their observed channel time share is too large [66]. Choi *et al.* address unfairness among downlink TCP flows by letting clients delay TCP acknowledgments and adapt the TCP advertised window [22].

A number of black box solutions have also appeared. Cai *et al.* resolve fairness among wireless flows using the wired router that is upstream of one or more APs [18]. On detecting unfairness, the wired router first throttles bandwidth to the affected APs to alleviate wireless congestion, then uses stochastic fair queuing. The work is restricted to downlink flows. Various black box-like solutions have been devised to address the imbalance between uplink and downlink TCP flows. Pilosof *et al.* propose to let an access point adjust advertised windows in TCP acknowledgments returned to senders [56], making several strong assumptions about TCP implementations (such as a data/ACK ratio of 1). Others let an AP compensate the imbalance between uploaders in aggregate vs. downloaders in aggregate and do not distinguish individual uploaders [14, 36, 45, 78].

The time-based regulator (TBR), our main point of comparison, is based on a black box approach. To control clients, it uses a token bucket scheme that schedules frame transmissions based on how much airtime a client has used [67]. While we adopt the idea of throttling TCP uploaders by rate limiting their downlink traffic, we have shown that TBR does not effectively control clients, since it lacks a means to estimate client demand.

3.7 Summary

In this chapter, we address the following fairness issues in 802.11 using a black box approach. First, by default 802.11 gives disproportionate advantage to uploading clients at the expense of downloading clients. Second, selfish uploaders may in fact increase their advantage over downloaders to an arbitrary extent. Third, poorly performing clients may degrade aggregate performance of an 802.11 channel. We propose FairEst, a black box approach to enforcing fairness, in which a wireless provider uses its access point to control unmodified clients by controlling their rate of airtime usage. FairEst is robust against selfish uploaders but relies on the assumption that uploaders depend on some form of feedback in downlink traffic (such as is the case for TCP). Existing work that uses a black approach suffers from one or more of the following limitations: it ignores uploaders, treats all uploaders as a single aggregate, or does not effectively estimate demand of uploaders, preventing it from achieving fairness for clients that are highly sensitive to other load on the channel. In other words, we claim that in previous work, an access point has insufficient control over uploading clients to enforce fairness, even under the assumption that uploaders use TCP.

By contrast, FairEst distinguishes among multiple uploaders and estimates a client's demand and airtime function using a simple linear model. While the linear model provides only an approximation of a client's airtime function, FairEst is able to correct for the model's inaccurate predictions and, in practice, achieves fairness on average. FairEst's control interface to clients consists of two parts: IP forwarding (delaying or dropping IP packets during forwarding) and monitoring 802.11 packets transmitted to or from the access point. This interface is highly portable and protocol feasible and has sufficient control over uploaders that satisfy our assumption of depending on feedback in downlink traffic. A current limitation of FairEst is that it reduces channel utilization to handle clients whose airtime function is insensitive to other load on the channel. In future work, we plan to address this limitation by explicitly detecting insensitive clients.

Chapter 4

Intelligent route service control point

The main claim of this dissertation is that the black box approach is applicable to a wide range of problems (Chapter 1). In Chapters 2 and 3, we apply a black box approach to improving efficiency and fairness, resp., of 802.11. In this chapter, we turn to a very different protocol, the Border Gateway Protocol (BGP), the main inter-domain routing protocol used among and within ISP networks to determine the flow of traffic traversing multiple ISP networks. Specifically, we address the problem of how a large ISP may control routing in its network (*route control*) in order to expand the range of services it can offer to customers, despite inherent limitations of BGP. In addition to tackling a different underlying protocol from previous chapters, the problem discussed in this chapter introduces the following challenge: due to the large number of routers contained in an ISP's network, any solution to the problem of route control must be highly scalable. In this chapter, we demonstrate that a black box approach can indeed be used to fully and efficiently control route selection in an ISP that operates a large, multi-vendor router base.

In Section 4.1, we explain why route control is necessary in modern ISPs. Specifically, we argue that modern ISPs need to be able to make real-time routing decisions based on network performance conditions, in order to offer competitive, customer-oriented services, such as load-balancing and low latency path selection. Section 4.2 provides background of the current routing architecture and practices and gives a concrete example of a route control objective that cannot be satisfied using current practices.

In Section 4.3, we present the design of IRSCP, a routing architecture that, based on a black box approach, gives a provider route control by controlling its existing routers. We describe our prototype implementation of IRSCP in Section 4.4 and demonstrate through experimentation that our implementation is capable of managing the routing load of a large Tier-1 ISP in Section 4.5. We evaluate our design and implementation based on our black box criteria in Section 4.7.

4.1 Motivation

Given the best-effort communication model of the Internet, routing has historically been concerned with connectivity; that is, with finding a loop-free path between endpoints. Deviations from this default behavior normally involve policy changes at slow time scales to effect business and network management objectives [17]. Within a particular network (or autonomous system), routing is governed by a standardized, fairly straightforward route selection algorithm, the *BGP decision process*, that tries to ensure consistent (*e.g.*, loop-free) decision making between the routers in the network, while respecting the network operator's policies.

Networked applications and traffic engineering techniques have evolved, however, placing increasingly demanding requirements on the routing infrastructure. For example, applications such as VoIP and online gaming may be highly sensitive to the characteristics of a chosen data path [19, 20, 53]. Indeed, a number of studies have shown that non-default Internet paths can provide improved performance characteristics [7, 9, 60], suggesting the potential benefit of making routing aware of network performance conditions [27]. Additionally, today's network operators occasionally need to restrict the traditional any-to-any connectivity model of the Internet to deal with DDoS attacks. Finally, there are cases where the BGP decision process is at odds with provider and/or customer goals and may, for example, lead to unbalanced egress links for customers that are multi-homed to a provider [72].

In order to satisfy these demands, modern ISPs need the ability to control routing in their networks (*route control*) that is (i) fine-grained, (ii) informed by information external to routing (such as network performance conditions), and (iii) applied at time scales much shorter than manual routing configuration changes and therefore implemented by means of a *route control application*. Unfortunately, BGP does not provide adequate means for performing such online, informed, and fine-grained route control.

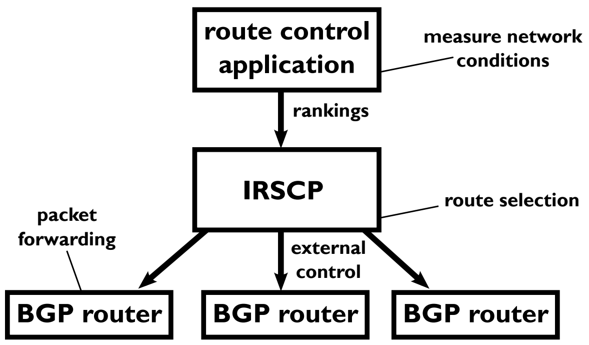


Figure 4.1: IRSCP architecture.

The tuning parameters BGP does provide are arcane and indirect. As result, operators and developers of route control applications manipulate BGP route attributes in cumbersome, vendor-specific router configuration languages at a low level of abstraction, frequently leading to ineffective or, worse, incorrect decisions [47].

In this chapter, we present the the design and implementation of a distributed Intelligent Route Service Control Point (IRSCP), a black box solution that allows an ISP to perform route control based on dynamic network conditions, using an intuitive abstraction based on a ranking of routes. Figure 4.1 shows the overall design of our IRSCP architecture. Central in our architecture is a distributed IRSCP platform (consisting of a set of IRSCP servers). In our architecture, the IRSCP platform, rather than routers, perform the route selection needed to implement the ISP's route control policy. The IRSCP platform uses a black box approach to externally control the ISP's unmodified routers, which remain responsible for forwarding packets according to the routes selected by IRSCP. IRSCP defines an interface that allows an ISP's route control application to guide IRSCP's route selection based on dynamic network conditions using an intuitive abstraction. In this dissertation, we are concerned with the black box aspects of route control, which are fully contained in the IRSCP platform. As such, we address the challenges in designing and implementing IRSCP, such that it provides an intuitive interface for route control applications, fully controls the ISP's router base, and is robust and efficient, as we discuss next.

Application-directed route control: IRSCP provides the mechanism that allows applications to dynamically manage connectivity. We allow an ISP's route control application to control route selection through an intuitive, vendor-independent interface. The interface is based on the abstraction of a *ranking* of routes for each router and destination. We introduce a modified BGP decision process, the *explicitly ranked decision process*, that incorporates the route rankings from route control applications to guide route selection in IRSCP. The key challenge to modifying the BGP decision process is to ensure that the resulting protocol retains (or improves on) BGP's robustness, scalability, and consistency properties. We present two simple constraints on the application-provided route ranking that ensure that IRSCP installs only safe routing configurations, even in the face of router failures or dramatic changes in the ISP's internal (IGP) topology.

Complete route control: IRSCP *fully controls* the forwarding behavior of routers in the ISP's network. As we argue in more detail later, this can only be achieved by having IRSCP communicate with routers in neighboring networks (in addition to routers in the ISP's own network), giving IRSCP *full visibility* of all routes available to the network. While RCP (Route Control Platform) [16] also uses the idea of controlling routers from a server, it does not implement full visibility and thus cannot provide the degree of control needed to implement application-directed route control.

Distributed functionality: In order to realize an architecture in which IRSCP has full control of route selection, we must address two further challenges. First, because all routers in the IRSCP-enabled network rely on the IRSCP for routing decisions, the IRSCP infrastructure must be highly robust. Second, IRSCP must be sufficiently scalable to control the large number of ISP routers and to learn routes from an even larger number of routers in neighboring networks. To address these concerns, we partition and distribute the IRSCP functionality across a number of servers while still ensuring consistent decision making for the platform as a whole.

4.2 The limitations of current practice

In this section we provide a brief overview of routing and forwarding practices in a modern MPLS-enabled ISP network. We then describe their limits using an example scenario based on load balancing.

4.2.1 Inter-domain routing in a modern ISP

Figure 4.2(a) shows a simplified view of the physical infrastructure of an MPLS-enabled ISP backbone. The routers at the periphery of the ISP network connect to

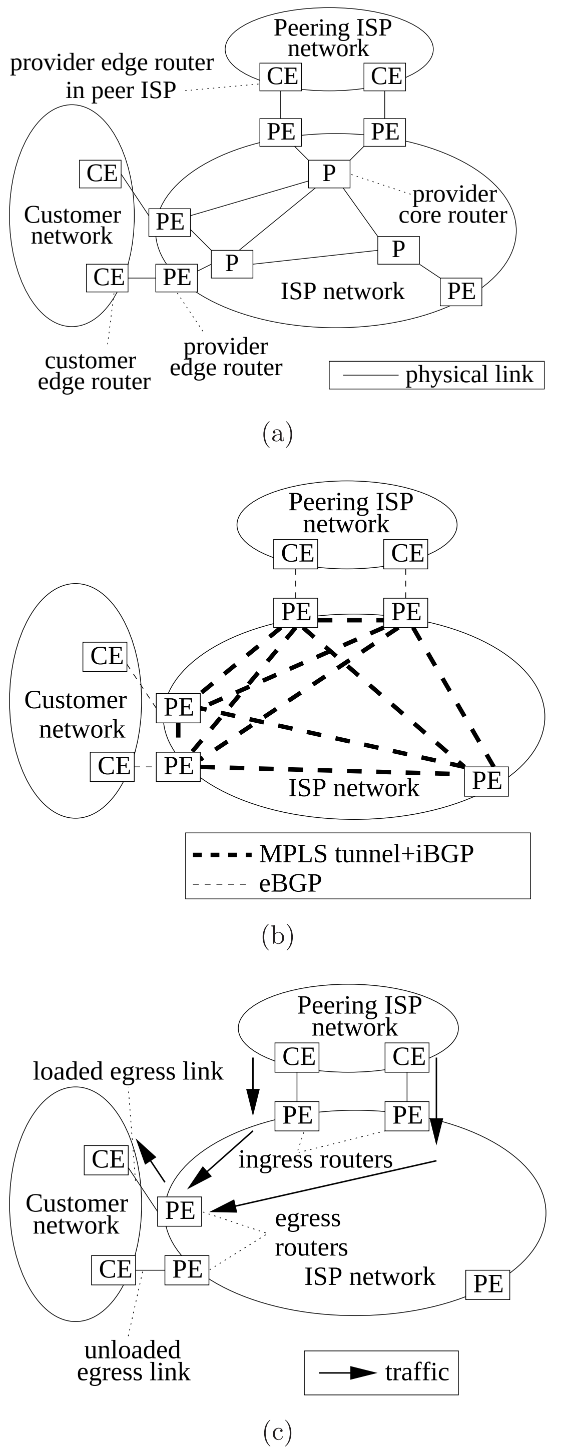


Figure 4.2: ISP routing infrastructure. (a) Physical connectivity. (b) BGP and MPLS. (c) Traffic ingresses and egresses.

Table 4.1: Left: the steps of the BGP decision process. Right: the steps of our *explicitly ranked decision process*. Steps 0-4 are identical and produce the *egress set*.

BGP Decision Process (Section 4.2.1)	Explicitly Ranked DP (Section 4.3.1)
0. Ignore if egress router unreachable in IGP 1. Highest local preference 2. Lowest AS path length 3. Lowest origin type 4. Lowest MED (with same next-hop AS)	(*same*)
B-5. eBGP-learned over iBGP-learned B-6. Lowest IGP distance to egress router B-7. Lowest router ID of BGP speaker	R-5. Highest explicit rank R-6. Lowest egress ID

other ISPs (called peers) and customers. These routers are called *Provider Edge (PE)* routers, and the routers that interconnect the PE routers are called *Provider Core (P)* routers. The customer routers connecting to PEs are called *Customer Edge (CE)* routers. For simplicity we also use *CE* to represent *peer* routers that connect to the ISP. BGP allows an ISP to learn about destinations reachable through its customers and peers. Typically every PE maintains BGP sessions with its attached CEs, and also with other PEs in the ISP network; the former are known as *eBGP* (external BGP) sessions and the latter as *iBGP* (internal BGP) sessions, as shown in Figure 4.2(b). When a PE router receives a route through its eBGP session, it propagates the route to other PEs through iBGP sessions, allowing every PE to learn how to reach every peer or customer network. The path traffic follows between *ingress* and *egress* routers is determined by another routing protocol known as an interior gateway protocol (IGP), such as OSPF. In an MPLS network, tunnels are established between all PEs (see Figure 4.2(b)) based on IGP, obviating the need to run BGP on P routers. Therefore, in the remainder, we will only be concerned with PE and CE routers.

A PE usually receives more than one egress route for a given destination and must run a route selection algorithm called the *BGP decision process* to select the best route to use for data forwarding. The BGP decision process (shown in the first column of Table 4.1) consists of a series of steps. Starting with the set of routes available to the PE, each step compares a set of routes and passes the most preferred routes to the next step while discarding the remaining routes. Steps 1–4 compare routes in terms of *BGP attributes* attached to the routes, while steps 0 and B-6 consider the IGP information associated with the egress PE of the route. We call the set of routes remaining after Steps 0–4 the *egress set*. Steps B-5 and B-6 are responsible for what is called *hot-potato routing*, *i.e.*, forwarding traffic to the nearest (in terms of IGP distance) egress PE in the egress set. Step B-7 is a tie-breaker that ensures that the PE always ends up with a single best route.

Network operators may configure *policy* statements on PEs to filter undesired routes or modify attributes of routes so as to influence the BGP decision process [17]. Policy can be applied on accepting routes from neighbors (*import policy*) and before sending routes to neighbors (*export policies*). Typically ISPs apply policies only on eBGP sessions and rarely on iBGP sessions.

4.2.2 Congestion aware load balancing

We use Figure 4.2 to highlight a specific problem introduced by BGP's hot-potato routing thereby motivating the need to enhance route selection with fine-grained application-directed route control.[1] We assume that the customer shown in Figure 4.2 has multihomed to the provider network for the purpose of improving redundancy, and so the same destinations are reachable via both links. (This is common practice for data centers and other customer networks with high availability requirements.) All PEs in the provider network therefore have two possible routes to reach the customer network. Assume further that most of the traffic destined to the customer network enters the provider network from the peering ISP network. Assuming unit IGP distances for each internal provider link in Figure 4.2(a), the two ingress PEs at the top of the figure prefer the route via the top egress PE connected to the customer network (Figure 4.2(c)). This leads to an imbalance in the load on the two egress links, with the top egress link carrying all (or most) of the traffic, which in turn may result in congestion.

In practice the customer or ISP may prefer the ISP to load-balance traffic on the two egress links while still considering the IGP distance between ingress and egress. To accomplish such a routing policy, several requirements need to be met. First, the ISP needs to be able to instruct each ingress PE to send traffic for a particular customer towards a different egress link. In other words the ISP needs the ability to not only specify per-destination, but also *per-PE* routing decisions. Second, such routing decisions need to be a function not only of routing information (*e.g.*, available routes and IGP

[1]A similar problem is introduced by *cold-potato* routing based on the BGP MED attribute (Step 4 in Table 4.1).

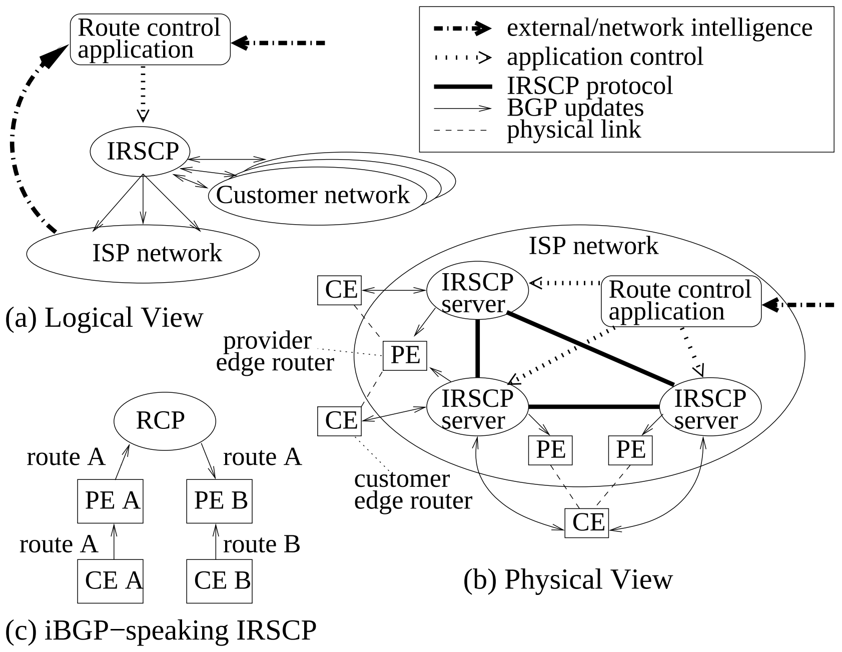


Figure 4.3: The IRSCP architecture in detail.

distance), but also of network performance conditions (*e.g.*, load on egress links). Third, since routing information and network performance conditions vary with time, routing decisions need to be made automatically, at real-time.

Unfortunately, the only way to implement per-destination, per-PE routing decisions in BGP is by manipulating BGP attributes on multiple PEs through vendor-specific configuration languages. Furthermore, BGP was not designed to take input from dynamic conditions other than routing information. By contrast, in IRSCP this type of route control is implemented by basing the decision process on the input from a load-balancing route control application run by the ISP, which takes into account IGP path distance, as well as offered ingress load and capacity of egress links. Using IRSCP, the route control application controls route selection for each PE independently and, in this example, instructs the two ingress PEs to each send traffic to a different egress PE. We call this *fine-grained, application-directed route control.*

4.3 IRSCP architecture

In this section we describe a black box approach to the design of a distributed Intelligent Route Service Control Point (IRSCP) that implements fine-grained, application-directed route control. Since an ISP typically operates many hundreds of routers, IRSCP treats each PE router as a black box, allowing an ISP to deploy route control for its existing router base. Below, we motivate the design of IRSCP using the *portability* and *sufficient control* black box criteria from Chapter 1. (We defer a discussion of *protocol feasibility* to Section 4.7.) Since PE routers are typically supplied by multiple vendors, each of which defines its own router configuration language, it is important that IRSCP uses a portable interface to control routers. Fortunately, each PE router is a BGP router, and as such, IRSCP is able to control PE routers using the BGP protocol itself. Specifically, IRSCP establishes an iBGP peering session with each PE and sends BGP updates to the PE to control how the PE forwards packets.

However, establishing an iBGP session with a PE is not in itself sufficient to give IRSCP full control over a PE router. For IRSCP to have full control, the PE must peer *only* with IRSCP. Therefore, it is IRSCP, rather than each PE, that establishes eBGP peerings with CE routers in neighboring networks, as shown in Figure 4.3(a). We use Figure 4.3(c) to emphasize the importance of having IRSCP peer with neighboring networks in order to establish full control over routers and routing. The figure shows an alternative design in which IRSCP only speaks peers with the ISP's PEs (similar to RCP [16]). For clarity we refer to such an iBGP-speaking IRSCP as RCP. RCP exchanges routes with PEs using iBGP and never communicates with CEs directly. Suppose that RCP sends an iBGP update to PE B containing route A. To implement full route control, the update must override any routes that PE B receives from other CEs (CE B). However, this implies that PE B never sends alternative routes (route B) to RCP unless route A fails (or changes its BGP attributes). RCP is thus deprived from using any but the first route it learns. It follows that an iBGP-speaking IRSCP restricts the ability to exercise full control.

To summarize, IRSCP receives BGP routes from CE routers through eBGP, executes a per-PE decision process to determine what routes each PE should use, and sends the resulting BGP routes to the PEs through iBGP, thereby fully controlling them. IRSCP also copies an update to each CE attached to a PE (again through eBGP) that corresponds to the routing decision that was made for the PE. To a CE router, IRSCP behaves just like a BGP router, therefore customers and peer networks need not be aware that their CEs peer with IRSCP rather than with BGP routers.

While IRSCP is responsible for implementing the mechanisms to select routes and control PE routers, our ultimate goal is for an ISP's route control application to be in charge of routing. Therefore, IRSCP is in turn controlled by a route control application, using external information (such as network performance measurements) to guide route selection in IRSCP, as shown in Figure 4.3(a). We define a powerful interface between IRSCP and the application that effectively permits the application to arbitrarily assign egress routes to PEs. In Section 4.3.1, we describe the IRSCP-application interface in detail, as well as IRSCP's *explicitly ranked decision process* that combines input from the application with available BGP learned routes in order to make routing decisions. While the IRSCP-application interface is powerful, it also brings with it the danger that the application's input might lead to forwarding anomalies, such as forwarding loops. In Section 4.3.2, we discuss how IRSCP protects itself from forwarding anomalies by enforcing two simple constraints on the application's input.

All of the above ensures that the route control application and IRSCP fully control the forwarding behavior of each PE router. However, full functional control in itself does not satisfy the *sufficient control* requirement to achieve an ISP's route control objectives. For sufficient control, IRSCP must be scalable and robust enough to control the many hundreds of PE routers in the ISP's network efficiently and reliably. In Section 4.3.3 we address this challenge by designing IRSCP as a distributed platform, consisting of multiple IRSCP servers that collectively perform route selection, as shown in Figure 4.3(b).

4.3.1 Explicitly ranked decision process

IRSCP implements two types of decision process: the normal BGP decision process and the *explicitly ranked decision process*. Both perform route selection on behalf of individual PEs and so are defined on a per-PE basis. The BGP decision process is used for the subset of destinations, *unranked prefixes*, for which the customer, ISP or route control application has determined that conventional hot-potato routing can be used. For the remaining *ranked prefixes*, the route control application creates a preference ranking of egress routes for each ingress router, the selection of which is realized in IRSCP by the explicitly ranked decision process.

In our architecture the route control application tells IRSCP which routers should use which egress routes based on routing information, traffic load measurements, etc. As a general principle IRSCP follows the directives of the application, except when doing so severely impairs connectivity. An instance of this principle is that we let the application specify a *ranking* of egress links, *i.e.*, egress links ranked by preference, rather than a fixed assignment of egress links to routers. (Each egress route corresponds to only one egress link, and we therefore use the terms egress link and egress route interchangeably.) Using a ranking accommodates unavailability of egress routes. For example, if the top-ranked egress route is unavailable, the next-ranked egress route may be selected. The application specifies the ranking on a per-destination, per-router basis.

We construct our decision process for ranked prefixes (Table 4.1) by adopting Steps 0–4 of the BGP decision process and then apply the explicit ranking instead of performing hot-potato routing, followed by a tie-breaker in Step R-6.[2] This ensures that the explicitly ranked decision process respects BGP attributes such as AS path length (Step 2) and that it takes reachability of egress routers into account. In principle, the explicit ranking can be applied at any point in the decision process, *e.g.*, it may override earlier steps of the decision process. However, we leave exploring the extent to which we may safely override or replace BGP attributes to future work.

We explore an example of the explicitly ranked decision process by considering the scenarios shown in Figures 4.4(a) and (b). In this example a single IRSCP server runs the decision process for every PE in the ISP's network. We examine the execution of the decision process for PE A in Figure 4.4(a). First the IRSCP server receives all routes for the given prefix: $E-C$, $F-C$ and $G-D$. (We refer to each route using its *egress ID*, the pair of (CE,PE) routers incident on the egress link for the route.) Next, the explicitly ranked decision process for PE A executes Steps 0–4 and in Step 2 eliminates egress route $F-C$ based on the longer AS path length. (We assume that the routes otherwise have identical BGP attributes.) The result is the egress set $\{E-C, G-D\}$. In Step R-5 the decision process applies the explicit ranking for PE A to the egress set. Since the top-ranked egress link $E-C$ is present in the egress set, the decision process selects this route for PE A. Similarly, the decision process selects route $E-C$ for PE C, and route $G-D$ for PEs B and D, resulting in the forwarding behavior shown in Figure 4.4(b). An important observation is that Steps 0-4 are identical for all PEs.

[2] In practice we also add a tie-breaker on 'IRSCP ID' at the end of the explicitly ranked decision process, based upon the IRSCP server that receives the route on an eBGP session.

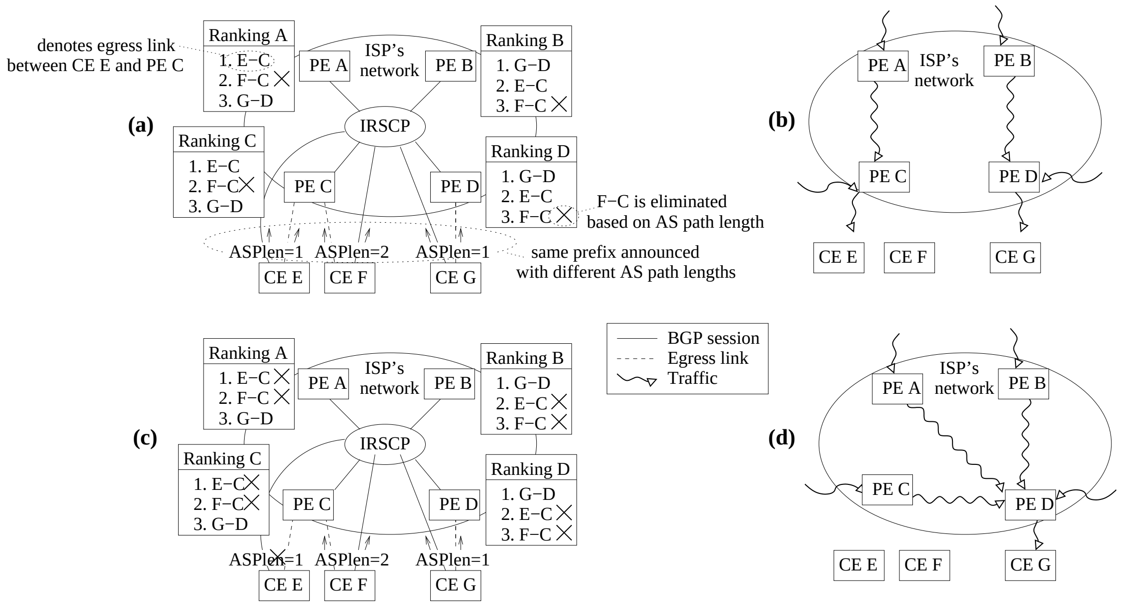


Figure 4.4: Example of ranking and its effect on forwarding. The application (not shown) has provided IRSCP with the explicit rankings indicated. Each ranking is a list of egress IDs. CEs E, F and G announce routes for the same prefix (with different AS path lengths) through their eBGP sessions with IRSCP. (a) Initial scenario. (b) Forwarding behavior for (a). (c) CE E withdraws its routes. (d) Resulting forwarding behavior.

Therefore the decision process for any PE computes the same egress set.

Outdated rankings

Ideally, the input from the application to IRSCP continuously reflects the current state of the network. In practice, however, IRSCP, being an active participant in BGP, is in a better position than the application to respond instantly to changes in routing state such as BGP route attributes (using Steps 0–4 of the decision process), IGP distances[3], and the availability of BGP routes (discussed below). IRSCP must therefore adapt the rankings from the application (based on possibly outdated routing information) to current routing information, as follows.

Between the time that the application sends the rankings to IRSCP and the time that the explicitly ranked decision process runs, new egress routes may be announced and old routes may be withdrawn. Until the application updates its rankings, IRSCP must accommodate discrepancies between the available routes assumed when the application

[3]The details of IRSCP's handling of IGP distance changes are discussed in Appendix A.

creates the rankings and the actual available routes. An instance of a withdrawn egress route is illustrated in Figures 4.4(c) and (d), in which CE E withdraws egress route $E-C$, and the egress set changes to $\{G-D\}$. As a result, the decision process changes its selection for PEs A and C to $G-D$ and all traffic egresses through PE D (Figure 4.4(d)). In other words, a ranking specifies not only the desired routing for the PE in the absence of failure, but also the desired fail-over behavior that the PE should adopt.

Conversely, if new egress routes are advertised, IRSCP appends them to the end of the explicit ranking (in order of egress ID) until the application is able to provide a revised ranking (Steps R-5 and R-6). Alternatively, the application may *prevent* IRSCP from appending routes in this manner. For example, the application may wish to restrict the set of egress routes of a particular customer to a fixed set, thereby preventing some forms of prefix hijacking. We define a 'virtual' *black-hole egress route*, which is part of every egress set and (conceptually) sinks traffic directed to it. We also define a corresponding *black-hole egress ID*, which an application can include as part of a PE's ranking. If the explicitly ranked decision process for a PE selects the black-hole egress route, the IRSCP server does not send a route to the PE (or its attached CEs), thus making the destination unavailable through that PE. (Step R-6 of the explicitly ranked decision process considers the value of the black-hole egress lower than any other egress ID, effectively ignoring it.)

Other IRSCP applications

By opening the decision process to external input, IRSCP enables a class of applications in which routing is controlled based on information external to BGP. An example of an application that uses external information (or in this case analysis) is Pretty Good BGP [41]. The authors propose a pragmatic approach to BGP security by which suspicious routes are quarantined for a certain period of time before being considered for selection. In this case knowledge about the historical availability of routes (even if the routes were not selected) is important to determine whether a route is potentially hijacked. Further, IRSCP provides a mechanism (the black-hole egress route) by which routers can be prevented from selecting suspicious routes until deemed safe by the application.

Another example in this category is the load-balancing application described above, which makes use of network conditions inside the IRSCP-enabled network to in-

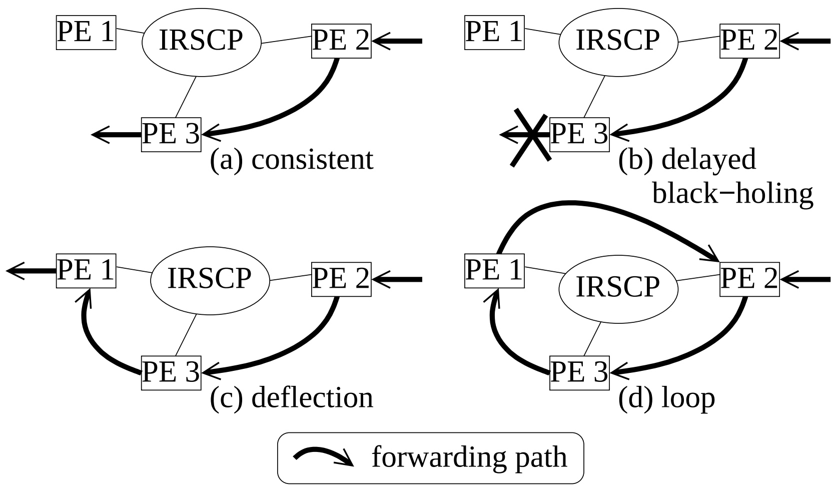


Figure 4.5: Forwarding anomalies. In all cases the decision process for PE 2 has selected an egress route through PE 3 as the best route for some destination. (a) The decision process for PE 3 has selected a local egress route as best route, and therefore is consistent. (b) The decision process for PE 3 has not selected any route for this destination, thus traffic is black-holed. (c) The decision process for PE 3 has selected PE 1 as best egress route, resulting in a deflection. (d) The forwarding loop is a result of multiple deflections.

form route selection. However, it is also possible to inform route selection with network conditions external to the IRSCP-enabled network. For example, Duffield *et al.* [27] explore the possibility of using measured path conditions to select between various alternate paths leading to the same destination. Based on these measurements, the network may then route traffic that is sensitive to adverse network conditions along paths more favorable than those that default BGP would be capable of finding (to the extent permitted by the policies encoded in BGP attributes).

The fact that IRSCP has full route visibility also simplifies a number of route monitoring applications. For example, auditing applications that ensure that peers abide by peering agreements require full visibility of routes advertised to a network [55]. Similarly, ‘what-if’ network analysis [30] becomes more straightforward if the application has access to the routes that were available in the past. IRSCP’s ability to use external input to inform route selection, however, is its key advantage.

4.3.2 Consistency of rankings

The concept of application-provided explicit rankings permits a route control application a great deal of flexibility. However, it also introduces the possibility of IRSCP executing the decision process in an inconsistent manner for different PEs, which can lead to forwarding anomalies. Figure 4.5 depicts the forwarding anomalies that may result: delayed black-holing, deflection, and forwarding loops. A packet is said to be deflected if a router on its forwarding path chooses to forward to a different egress router than the egress router previously selected by a router upstream on the forwarding path [34]. In an MPLS network deflections only occur at egress routers. We wish to prevent deflection for two reasons. First, given the existence of shortest-path MPLS tunnels between two PEs, forwarding through an intermediate PE is suboptimal (Figure 4.5(c)). Second, multiple deflections can lead to a forwarding loop (Figure 4.5(d)). Similarly we wish to avoid delayed black-holing as in Figure 4.5(b) since it is wasteful to carry traffic through the network only to have it dropped. If the intent is for the traffic to be dropped, it should be dropped on ingress (*i.e.*, at PE 2).

Ultimately, the *correctness* of the rankings is specific to the application. However we consider *consistency* to be a minimum standard of correctness for any route control application and therefore define a set of per-destination constraints on any set of rankings provided by an application. Enforcing these constraints (by an application or by a resolver in a pluggable route control application) ensures that the explicitly ranked decision process is *deflection-free*, *i.e.*, free of deflections and delayed black-holing. (In Appendix A we show that the BGP decision process as implemented by IRSCP is deflection-free.)

We define the operator $<_r$ as: $e_1 <_r e_2$ iff in the explicit ranking for router r egress link e_1 is ranked above egress link e_2. For example, for PE A in Figure 4.4(a), we have $E - C <_A F - C$ and $F - C <_A G - D$. We noted earlier that Steps R-5 and R-6 of the explicitly ranked decision process effectively extend each explicit ranking to include all egress routes. We say that $e \in <_r$ if e is in the explicit ranking for r. For example $E - C \in <_A$, but if some new egress D-H appeared then $D - H \notin <_A$. The constraints on explicit ranking are as follows.

Definition: *Ranking-Consistent-1:* The set of egress routes appearing in each router's explicit ranking is identical.

Definition: *Ranking-Consistent-2:* For each router r and all egress links e_1, e_2:

if $e_1 <_r e_2$ then $e_1 <_{pe(e_1)} e_2$, where $pe(e)$ is the PE incident on e.

The rankings shown in Figure 4.4(a) clearly satisfy Ranking-Consistent-1: all rankings contain the same egress links. They also satisfy Ranking-Consistent-2. For example, checking the ranking for PE B we see that (1) $G-D <_B E-C$ and $G-D <_D E-C$, (2) $G-D <_B F-C$ and $G-D <_D F-C$, (3) $E-C <_B F-C$ and $E-C <_C F-C$. In Appendix A, we prove that if the explicit rankings given to an explicitly ranked decision process satisfy Ranking-Consistent-1 and Ranking-Consistent-2, then the explicitly ranked decision process is deflection-free.

Essentially, the ranking abstraction is able to describe a preferred egress link for each PE and per-PE fail-over behavior such that traffic does not get deflected. It is powerful enough to express any consistent assignment of egress routes to routers. However, the constraints do not permit failing over from one arbitrary consistent assignment to another. For example a given set of rankings that ranks egress link e_1 highest for PE A cannot fail over in such a way that egress link e_2 is assigned to PE A, unless e_1 fails.

4.3.3 Distributed IRSCP architecture

In order for IRSCP to maintain sufficient control of routers to achieve the ISP's route control objectives, it needs to be robust[4] and scalable. Therefore, though logically centralized from a route control application viewpoint, IRSCP is implemented as a distributed system—consisting of multiple IRSCP servers. If we designed IRSCP as a single centralized server, failure or partitioning away of that server would leave every PE in the network steerless and unable to forward traffic correctly, since in BGP a session failure implicitly causes BGP routes announced through the session to be withdrawn. In IRSCP we tolerate the failure of an IRSCP server by letting routers peer with multiple IRSCP servers. Furthermore, a centralized IRSCP faces a number of (overlapping) scalability challenges. First, a large Tier-1 ISP's routers collectively maintain many thousands of BGP sessions with routers in neighboring networks, something that no current BGP implementation is able to support by itself. Second, IRSCP makes independent routing decisions for each of the hundreds of PEs within the ISP. Third, IRSCP must store the combined BGP routes received from CEs and originated by the network, and it must process updates to these routes.

As shown in Figure 4.3, we choose to partition the IRSCP workload by letting

[4]A full discussion of fault tolerance would take us too far afield and is therefore left for Appendix A.

each IRSCP server peer with a subset of PEs and CEs, thereby addressing the first two scalability concerns. Our architecture still requires each IRSCP server to store all BGP routes and process the corresponding updates; however, we show in Section 4.5 that this aspect of scalability need not pose a problem for current server hardware.

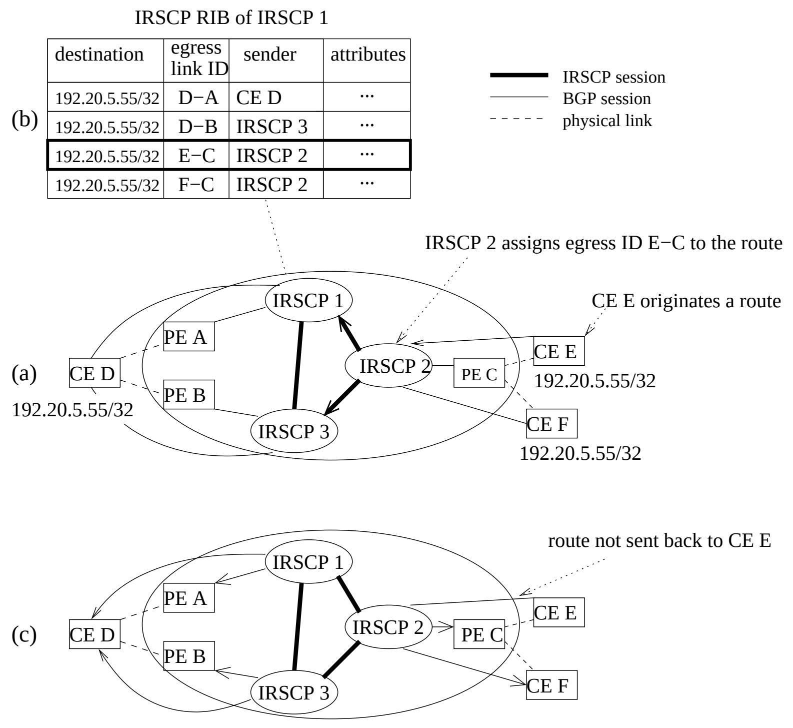
In performing the per-PE decision process, an IRSCP server needs to take into account the position of that PE in the IGP topology. To maintain consistency of IGP routing state, each IRSCP server runs an IGP viewer and has the same global view of the IGP topology [16]. Due to the partitioning of work in our distributed architecture, each IRSCP server needs to perform shortest-path calculations only for the set of PEs for which it makes BGP routing decisions rather than for the network as a whole. The IRSCP servers further ensure consistency by exchanging *all* BGP updates among each other. Comparing this solution with BGP route reflection [13], the most important difference is that a route reflector selects a *single route* as best route for each destination (using the traditional BGP decision process) and only makes that route available to other routers and route reflectors. As a result different routers and route reflectors may observe a different set of available routes, which in turn has led to non-deterministic or divergent behavior in BGP, *e.g.*, 'MED oscillation' [33, 49].[5] Basu *et al.* [12] show that exchanging all routes selected by Steps 0-4 of the decision process is sufficient to prevent non-determinism and divergence in iBGP; IRSCP exchanges a superset of this set.

Thus we propose a simple and elegant solution to distribution: a full mesh in which all IRSCP servers distribute all routes. Architecturally, this solution is similar to an iBGP infrastructure in which BGP routers are fully meshed, an architecture that was eventually replaced by route reflectors due to scalability problems. However, there are two differences between these two architectures. First, the number of IRSCP servers is small compared to the number of routers. Second, being based on commodity technology rather than router technology, we expect IRSCP servers to better keep pace with the technology curve than routers have [39].

IRSCP protocol and RIB

The *IRSCP protocol* (Figure 4.3(b)) is responsible for ensuring route exchange among IRSCP servers and is implemented as a straightforward extension to BGP. Each pair of IRSCP servers maintains a TCP-based *IRSCP session* through which they ex-

[5] MED oscillation continues to be observed in the Internet [77].



IRSCP RIB of IRSCP 1

destination	egress link ID	sender	attributes
192.20.5.55/32	D−A	CE D	...
192.20.5.55/32	D−B	IRSCP 3	...
192.20.5.55/32	E−C	IRSCP 2	...
192.20.5.55/32	F−C	IRSCP 2	...

Figure 4.6: IRSCP route propagation example.

change incremental updates in the form of advertisements and withdrawals of routes. At session startup the IRSCP servers exchange advertisements corresponding to all known BGP-learned routes, similar to traditional BGP sessions. When an IRSCP session disconnects, all routes exchanged previously on the session are implicitly withdrawn. When an IRSCP server learns a route from a CE router, it sends the route to all other IRSCP servers through the IRSCP sessions.

Figure 4.6 shows an example where the same destination prefix (192.20.5.55/32) is advertised by three different CEs connected to three different PEs. As shown in Figure 4.6(a), IRSCP 2 learns two routes to destination 192.20.5.55/32 through BGP, and it must send both instances to IRSCP 1 and 3. To distinguish several routes for the same destination, the IRSCP protocol includes in each update an identifier for the egress link to which the route corresponds. The *egress link identifier* (*egress ID* for short) is a (CE,PE) pair of the routers incident on the egress link. For example, from Figure 4.6 the routes learned through IRSCP 2 have egress IDs $E - C$ and $F - C$.

When an IRSCP server receives routes from BGP routers and from other IRSCP servers, it stores the routes in a routing information base (RIB), so that the routes are available to the decision process and for further propagation to routers and IRSCP servers. For example, the IRSCP RIB of IRSCP 1 shown in Figure 4.6(b) contains four entries for prefix 192.20.5.55/32. Each entry has fields for the destination prefix, the egress ID, the neighbor of the IRSCP server from which the route was received, and BGP attributes that belong to the route. In the example, CE D has sent routes to IRSCP 1 and 3, resulting in the first two entries. The last two entries correspond to routes sent to IRSCP 2 by CEs E and F.

Based on the RIB, an IRSCP server executes a decision process and sends a single route per destination (Figure 4.6(c)) to each attached BGP router. Since the IRSCP server only advertises one route per destination through each BGP session, the egress link ID is not needed (nor recognized by BGP) and so is stripped before sending the route.

4.4 Implementation

Since an IRSCP server uses the BGP protocol to control routers as black boxes, it is important that IRSCP is indistinguishable from a standard BGP router from the point of view of a controlled router. A significant part of the IRSCP server functionality is therefore identical to that of a BGP router, and we base our prototype implementation on an existing BGP router implementation: `openbgpd` version 3.9.

Figure 4.7 summarizes the control flow among the various components that our implementation uses to receive, store, process and send routes. Note that while an IGP viewer implementation exists for a previous centralized control platform [16], we have not yet ported it to our IRSCP prototype. In addition, BGP defines a rate limiting mechanism (*MinRouteAdvertisementIntervalTimer* [58]) that we have yet to implement for BGP sessions and adapt for IRSCP sessions. With the aid of Figure 4.7, we now discuss the implementation of a number of specific IRSCP functions in detail.

4.4.1 IRSCP-application interface

IRSCP exports an interface to route control applications, allowing them to control IRSCP's route selection process. Specifically, as shown in Figure 4.7, a route control application provides *explicit rankings* to IRSCP, which the IRSCP's explicitly ranked decision process then combines with routing information in order to make routing decisions. In our prototype implementation, we use a preliminary implementation of the

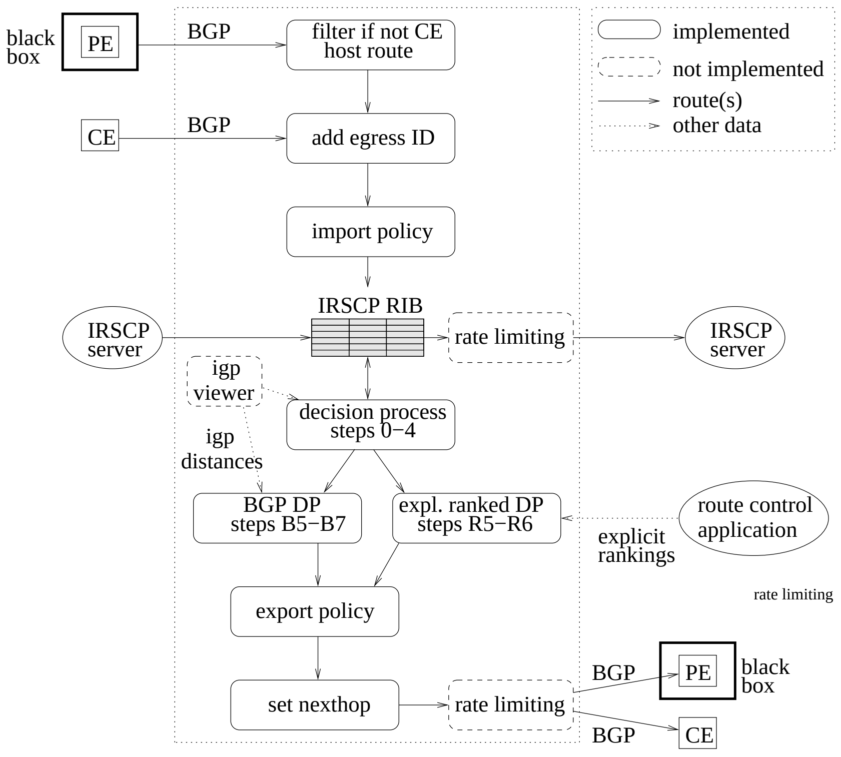


Figure 4.7: Control flow through an IRSCP server.

interface based on a configuration file that contains a number of `rank` statements, one for each set of prefixes that are ranked identically. The following is an example of a `rank` statement for two prefixes.

```
rank {
 prefix   { 3.0.0.0/8, 4.0.0.0/8 }
 pe       { 1.2.3.4, 5.6.7.8 }
 egresses { 11.12.13.14:15.16.17.18,
            19.20.21.22:23.24.25.26 }
 pe       { 101.102.103.104 }
 egresses { 19.20.21.22:23.24.25.26,blackhole }
}
```

The `pe` statements of a `rank` statement must contain every PE attached to the IRSCP server. Recall that in IRSCP each route carries an egress ID, specifying what egress link traffic for the destination of the route is to use when it exits the network (assuming the route is selected as best route). The `egresses` statement is the ranking of egress IDs to be used for the PEs in the preceding `pe` statement. Each egress ID consists of the IP addresses of the CE and PE incident on the egress link. In addition, the `blackhole` egress may be specified. Application rankings are stored in a per-PE Red-Black Tree of destination prefixes, where each prefix points to the ranked list of egress IDs for that PE and prefix. When a new ranking set arrives, the IRSCP server updates its ranking tree and reruns the decision process for affected destinations.

4.4.2 Decision process and IRSCP RIB

To control PE routers, an IRSCP server implements its own decision process that it runs on behalf of PEs, which accesses the IRSCP's routing information base (RIB). Compared to a BGP router implementation, performance is a concern in an IRSCP server implementation, since an IRSCP server must run a decision process for multiple PEs. In this section we consider the data structures used to implement the RIB and the time complexity to access the RIB as well as run decision processes.

The data structures used to implement the RIB (*IRSCP RIB* in Figure 4.7) are adapted from `openbgpd`'s implementation of the BGP RIB. `openbgpd` provides various indexes into the BGP RIB, the most important of which is a Red-Black Tree of destination prefixes, where each entry points to a list of routes (one route for each neighbor). To simplify the implementation we maintain this structure, despite the fact that the number of routes per prefix in IRSCP increases to one route per egress link. Our performance evaluation in Section 4.5 shows that the resulting increase in search time for a route in the RIB is not a practical concern, and therefore we leave further optimization as future work.

`openbgpd` maintains the per-destination list of routes in order of preference, based on pairwise comparison of the BGP attributes of routes according to the BGP decision process steps shown in Table 4.1. Thus each route update is linear in the number of routes for the destination, and the decision process itself amounts to no more than checking whether the egress router for the route at the head of the list is reachable (Step 0).

However, there are two problems with this algorithm for IRSCP. First, pairwise comparison of the MED value (Step 4) produces results that may be incorrect and, furthermore, dependent on the order in which the routes were inserted into the RIB, which in turn leads to potential inconsistency among the RIBs in different IRSCP servers. The underlying problem is that MED, being comparable only between routes from the same neighboring network, does not follow the 'rule of independent ranking' [33].

The second problem is that IRSCP defines a per-PE decision process, resulting in a different order of routes for different PEs. However, we note that Steps 0–4 of the (BGP or explicitly ranked) decision process (which compute the egress set) are independent of the PE, hence we compute these steps once for all PEs (*decision process steps 0–4* in Figure 4.7), storing the result in the RIB. Our implementation orders routes in the RIB using pairwise comparison based on Steps 0–3. Next, of the routes ranked highest so far, it examines each set of routes received from the same neighboring network and sets a flag on those that have highest MED value in such a set, thereby computing the egress set (Step 4). The algorithm used for Step 4 is similar to that used by the Quagga open source BGP stack (`www.quagga.net`) and runs in $O(n^2)$ time for n routes available for a prefix.

Next, the PE-specific steps of the decision process are executed using a pairwise comparison of routes in the egress set. In the case of the BGP decision process (*BGP DP steps B5-B7* in Figure 4.7), we break the tie in Step B-7 based on the egress ID of the route rather than the router ID (of the BGP router or IRSCP server that sent the route), since a neighboring IRSCP server may have sent multiple routes for the destination. In the case of the explicitly ranked decision process (*expl. ranked DP steps R5–R6* in Figure 4.7), the IRSCP server adds a black-hole route (with lowest possible egress ID) to the egress set and then retrieves the list of ranked egress IDs for the PE and destination (Section 4.4.1). When comparing the egress IDs of a pair of routes in the egress set, our implementation simply walks down the list of egress IDs until it finds the higher ranked of the two egress IDs. This runs in time linear in the number of routes times the size of the ranking for a prefix and PE, or, if all n prefix's routes appear in its ranking, in $O(n^2)$ time.

Following (re-)execution of the decision process for a given PE, the IRSCP server distributes its decision to the PE and to all CEs attached to the PE. Note that the IRSCP server does not need to run a separate decision process for a CE: as in BGP, the route sent to the CE is the same as is selected for the associated PE.[6] To prevent unnecessary updates from being sent to neighbors, BGP implementations typically remember the last route selected for each destination as the *active route*. Our IRSCP server implementation uses the same technique; however, instead of storing a single active route, it stores an

[6]Ignoring the fact that eBGP export policies may still be applied.

active route for each attached PE.

To evaluate scaling of the decision process in terms of the number of routing policies, we should consider not only the time needed to evaluate the ranking of a prefix (linear time, see above), but also the time needed to look up the ranking. Retrieving a ranking runs in time logarithmic in the number of ranked prefixes and (in our implementation) linear in the number of PEs per IRSCP server. This is similar to the time needed by a BGP implementation to look up a prefix in the routing table and retrieving peer state before sending updates to a peer.

4.4.3 Egress management

Based on the egress ID specified in each IRSCP route, IRSCP has to ensure that (a) BGP routers are instructed to forward the traffic towards and through the indicated egress link, and (b) the route is only used if the egress link is available. To implement (a) we use the *nexthop* BGP attribute of a route. This attribute tells a router to which *nexthop router* it must forward traffic when it uses the route. The IRSCP server sets the nexthop attribute (*set nexthop* in Figure 4.7) when it sends a route to a PE or CE in such a way that traffic passes through the egress link and uses the egress ID to determine what that nexthop should be. For example in Figure 4.4(a) and (b), IRSCP determines from egress ID $E - C$ that PE A must use PE C as nexthop, and PE C must use CE E as nexthop. In addition, a CE attached to PE A (not shown) is sent PE A as nexthop. The latter is not determined based on egress ID. Rather, as we discuss next, an IRSCP server associates each CE with a PE, and it is this PE that IRSCP sets as nexthop when sending to the CE.

When an IRSCP server is configured to form an eBGP session with a CE, part of the configuration identifies a PE with which to associate the CE: the PE to which the CE is physically attached through an egress link (*e.g.*, see Figure 4.3(b)). Apart from setting the nexthop router for routes sent to the CE (as discussed above), the IRSCP server uses the identity of the PE and CE to set the egress ID (*add egress ID* in Figure 4.7) on routes received from the CE.

To implement (b) above, the IRSCP server monitors the validity of the egress link and thereby of any routes received from the CE. Normally when two BGP routers exchange routes through a BGP session, the validity of the routes are preconditioned on the liveness of the session: if the session goes down, it is assumed that the physical links that carry the session's packets are unusable and any routes exchanged must be withdrawn.

However, in the case of a multihomed customer it is possible that the eBGP session's packets are routed through any of the customer's egress links.[7] Therefore an IRSCP server explicitly makes an eBGP session dependent on the correct egress link by checking whether an *egress status route* was received from the PE. The egress status route is a BGP route for the address of the CE, configured on the PE, and made dependent on the presence of the egress link. The IRSCP does not allow the eBGP session to be formed unless the egress status route for that session is present.[8] Note that in the IRSCP architecture presented in Section 4.3, an IRSCP server normally does not receive routes from PEs. The egress status route is an exception, and our IRSCP implementation guards against configuration errors by ensuring that only host routes are received from PEs (*verify egress route* in Figure 4.7).

4.4.4 Import and export policy

IRSCP provides a convenient interface for controlling the flow of traffic through an ISP's network. However it must also accommodate today's routing policies, *e.g.*, controlled distribution of routes to customer vs. peering networks, manipulating BGP attributes such AS paths and MED values, and filtering undesired routes from neighboring networks. Although ideally IRSCP should enable a centralized configuration of routing policy through an application, we leave this aspect of route control as future work. In the meantime, it is important for IRSCP to support today's style of routing policy configuration without introducing inconsistency into the decision making.

To ensure consistency we allow policy to be applied only after receiving a route from a CE (*import policy*) or before sending a route to a CE (*export policy*). Applying policy at these points can never hurt consistency, since the effects are identical to the CE applying a local export or import policy, respectively. We believe that the combination of IRSCP ranking and CE-import and CE-export policy enables virtually all policy in use today. For example, all import and export policies discussed by Caesar *et al.* [17] are configured for routes entering or leaving the network. One class of policies that we do

[7]In conventional BGP, it is considered good practice to only configure eBGP sessions between routers that share a common link, namely the link used by traffic for destinations exchanged on the session, and to restrict the session's packets to use that same link.

[8]Egress status routes also allow the large number CE IP addresses to be kept outside of the ISP's IGP. Some ISPs prefer to minimize the number of IP addresses carried by their IGP.

not allow are those that have a per-PE (as opposed to network-wide) effect. Examples of mechanisms that implement such policies are Cisco's weight and cost community [23, 59] attributes. A similar effect can be achieved in IRSCP using an explicit ranking.

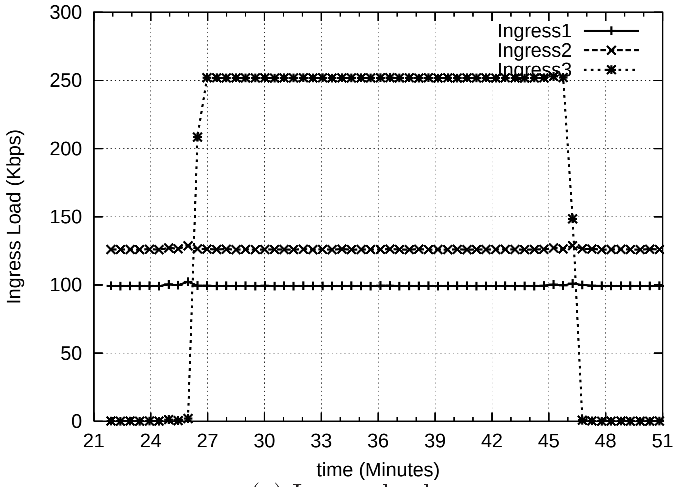
4.5 Evaluation

A non-black box approach to fine-grained, application-directed route control might be implemented by modifying the implementation of each PE to run its own explicitly ranked decision process and, as in standard BGP, maintain its own peerings with other PEs in the ISP's network as well as with CEs in other networks. By contrast, in the IRSCP architecture, an IRSCP server runs the decision process and maintains CE peerings on behalf of multiple PEs. As a result, one might expect that an IRSCP server implementation would require powerful and expensive hardware to keep up with an ISP's routing load. In Sections 4.5.2–4.5.4 we determine the feasibility of our approach by evaluating the extent to which our prototype implementation, while running on moderately equipped commodity server hardware (3.6-GHz Xeon, 4 GB), is able to keep up with the routing load of a large Tier-1 ISP. However, we start by presenting a functional evaluation in Section 4.5.1, demonstrating that an example load-balancing application is able to effect fine-grained route control using the ranking abstraction and our prototype IRSCP implementation.

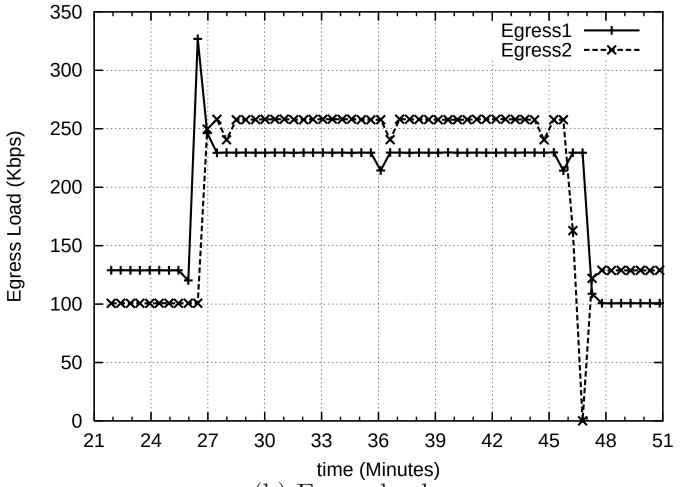
4.5.1 Functional evaluation

We verify the functionality of IRSCP with a testbed consisting of an IRSCP server, a load-balancing route control application, three ingress PEs, (Ingress1, Ingress2, and Ingress3), two egress PEs, (Egress1 and Egress2), a core router, and a number of hosts arranged into a source network and a destination network. The core router uses point-to-point links to connect to the PEs and we use GRE tunnels as the tunneling technology between the PEs. We assume a simple scenario in which the egress PEs attach to a customer who prefers its incoming traffic to be balanced on the egress links.

The load-balancing application performs SNMP GET queries on each PE to collect offered and egress loads. Based on the offered load, the application computes a ranking set that balances the load on the egress links. If the ranking set changes, the application generates an updated configuration and issues a configuration reload command. The hosts in the source network send constant rate UDP packets to the hosts



(a) Ingress load



(b) Egress load

Figure 4.8: Functional evaluation.

in the destination network. Figure 4.8(a) shows the offered load at the ingress PEs as measured by the application server.

The load at the egress PEs is shown in Figure 4.8(b). Before $t = 26$, the load at these two egresses is 'balanced" as a result of the initial ranking set at the IRSCP (*i.e.*, Ingress1 prefers Egress2, and Ingress2 prefers Egress1). At $t = 26$, the offered load at Ingress3 increases, and the application takes up to 30 seconds (the SNMP polling interval) to detect this change. It then realizes that the best way to balance the load is to send the load from Egress1 and Egress2 to one egress point, and the load from Egress3 to another. Therefore, it generates a new ranking set and sends it to the IRSCP server, which updates the ingress PEs. As a result, the load from Ingress3 goes to Egress2, and

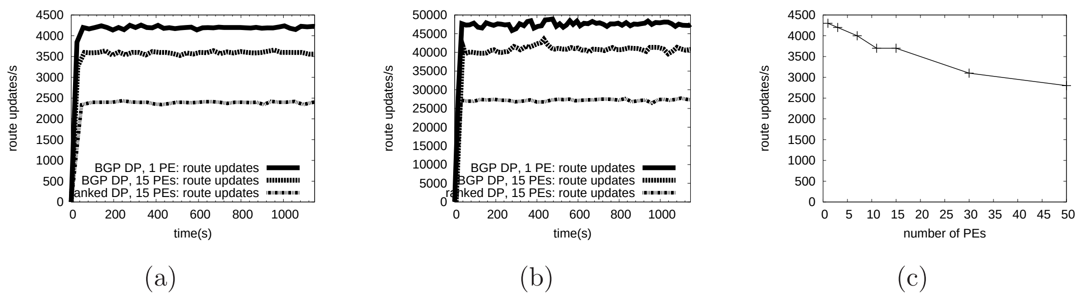


Figure 4.9: IRSCP server throughput. (a) Sustained input rate: from update generator to IRSCP server. (b) Sustained output rate: from IRSCP server to update receiver. (c) Input rate for varying number of PEs sustained for 40 seconds.

the load from Ingress1 and Ingress2 goes to Egress1. Similarly, at $t = 47$ the application generates a new ranking set and shifts Ingress1 and Ingress2's loads to different egresses.

4.5.2 Performance measurement testbed

We now turn to our performance evaluation of IRSCP, determining whether it can handle a Tier-1 ISP's routing load. Rather than emulating an entire ISP infrastructure, we run our tests on a single IRSCP server, loading it as though it were part of a hypothetical Tier-1 ISP network. Note that there is limited value in emulating an IRSCP-based infrastructure for the purpose of performance evaluation. Due to the fact that IRSCP servers exchange all routes, routing in an IRSCP infrastructure does not oscillate like it does in a BGP-based infrastructure [12]. (Whereas BGP routers and route reflectors exchange only their best routes, IRSCP servers exchange all eBGP-learned updates with each other, and so an IRSCP server does not change its route selections for a destination once it has received all such routes.) Therefore convergence time of an update is governed solely by communication latency and the processing latency of an update in an IRSCP server or a BGP router.

Figure 4.10(a) shows the Tier-1 ISP network modeled. Each of the n_{pop} POPs (Point-of-Presence, *e.g.*, a city) contains one IRSCP server and all IRSCP servers are fully meshed. Assuming IRSCP as a whole processes a combined BGP update[9] rate $rate_{all}$ from all CEs, then on average an IRSCP server sends roughly $rate_{all}/n_{pop}$ to each IRSCP server, namely the share of updates that it receives from CEs in its own POP. In other

[9]By 'update' we mean a route update, rather than an update *message*, which may contain several route updates.

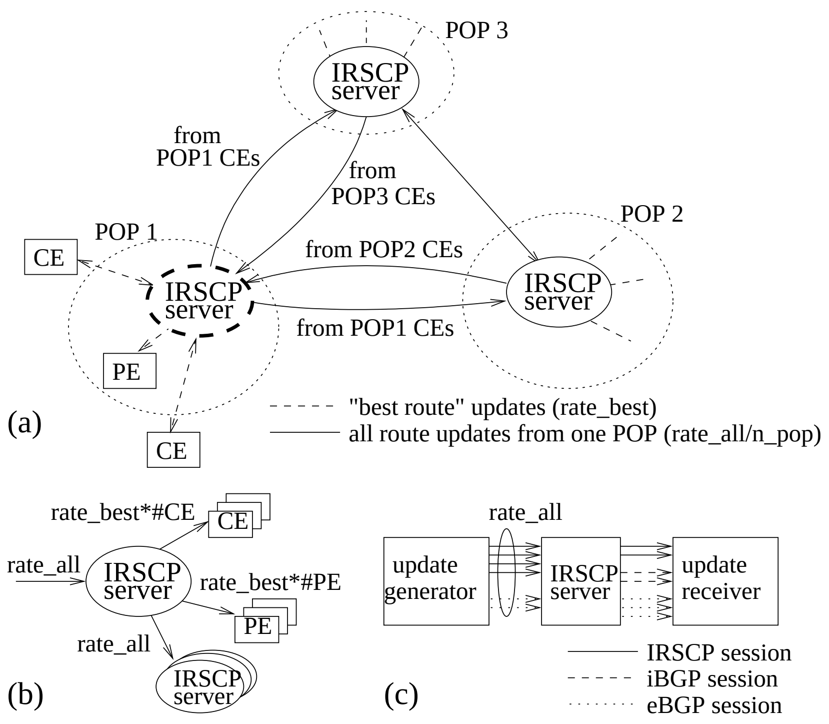


Figure 4.10: (a) Communication peers of IRSCP server under test. (b) Update rates into and out of the IRSCP server. (c) Experimental setup.

words, each IRSCP server sends and receives at a rate of about $\frac{n_{pop}-1}{n_{pop}} \cdot rate_{all} \approx rate_{all}$ on its combined IRSCP sessions (Figure 4.10(b)). The update rate that an IRSCP server sends to a PE or CE is governed by the output of best route selection based on the (BGP or explicitly ranked) decision process. We make the simplifying assumption that this rate $rate_{best}$ is the same for all PEs and CEs. From measurements taken in a Tier-1 ISP network we derive highly conservative ball park estimates for $rate_{all}$ and $rate_{best}$ (see next section).

Figure 4.10(c) shows our experimental setup, consisting of an update generator, the IRSCP server under test and an update receiver. The IRSCP server and update receiver are 3.6-GHz Xeon platforms configured with 4 GB of memory and a 1-Gb Intel PRO/1000MT Ethernet card. The update generator contains a 993-MHz Intel Pentium III processor, 2 GB of memory, and a 100-Mb 3Com Ethernet card. All three machines run OpenBSD 3.8.[10] The three hosts are connected through a Gb switched VLAN, and we monitor their communication by running `tcpdump` on a span port on a separate

[10]The machines are actually dual-processor, but OpenBSD uses only one of the processors.

host (not shown). The update generator and receiver are based on an IRSCP server implementation, with the update generator's implementation modified to send updates at a desired rate. We configure our setup to only permit updates to be sent from the generator to the IRSCP server and from there to the receiver.

After being loaded with a routing table of 234,000 prefixes, the update generator randomly picks prefixes from the routing table to produce the updates. To ensure that the IRSCP server's RIB contains a sufficient number of routes per prefix to choose from, the update generator and the IRSCP server maintain 26 sessions. Each time the update generator sends an update message it picks a different session through which to send the message (using round-robin). The message modifies a BGP attribute (the *aggregator* attribute) of the prefixes contained in the message.

The IRSCP server and the update receiver maintain 40 IRSCP sessions, 15 iBGP sessions, and 240 eBGP sessions, reflecting an ISP network consisting of 40 POPs, each of which contains 15 PEs and has sessions with 240 CEs. (Alternatively, this scenario reflects a network consisting of 20 POPs of twice the number of routers, but where each POP contains two IRSCP servers that assume responsibility for half the POP.) We need to ensure that each BGP and IRSCP session with the update receiver sends at a rate of $rate_{best}$ and $rate_{all}/n_{pop}$, resp. Each per-PE decision process in the IRSCP server prefers one of the 26 routes available. Only when the attribute of the best route is changed does the IRSCP server send an update on the iBGP sessions for that PE (and on the eBGP sessions for the associated CEs). By virtue of route selection in the IRSCP server each output BGP session receives a 1/26 fraction of $rate_{all}$, which turns out to correspond to $rate_{best}$ (see below). We configure one of the 26 input update-generator sessions as an eBGP session and the remainder as IRSCP sessions, ensuring a rate of $rate_{all}/26 > rate_{all}/n_{pop}$ on each output IRSCP session. Finally we instrument the IRSCP server and update generator to send at most three route updates per update message, reflecting the average number of updates per message observed at a route collector in the Tier-1 ISP's network.

We disabled updates to the kernel forwarding table as well as `openbgpd`'s support for kernel packet filtering, neither of which are required by IRSCP. In addition we prevented `openbgpd` from disabling Nagle's algorithm (TCP_NODELAY) on the peering sessions to achieve better throughput. Before starting each experiment we allowed the routing table to be propagated to the IRSCP server and the update receiver, through each of the sessions.

4.5.3 Workload used for evaluation

In order to place a reasonable workload on the IRSCP server under test we establish some ball-park upper bounds on the workload of an IRSCP server in deployment, using data collected from a large Tier-1 ISP network. Specifically we estimate upper bounds for the following two items: (a) number of prefixes and routes carried by an IRSCP server and (b) update rate sent and received by an IRSCP server to and from routers and other IRSCP servers. For (a) we need to count the number of routes and prefixes that are announced by neighboring (customer and peering provider) networks and customer routes originated by the ISP on behalf of customers. We count the number of routes contributed by peering provider networks using BGP routing table dumps taken from the PE routers that maintain BGP sessions with these peering providers. For routes contributed by (or originated on behalf of) customers (*i.e.*, the *customer routes*) we do not have access to corresponding BGP table dumps. However we do have access to *forwarding* table dumps of the PE routers connected to customers, which give us for each customer the set of destinations forwarded to the customer, *i.e.*, the customer's prefixes. In addition we have a per-customer list of interfaces through which the customer connects to the ISP. We make the conservative assumption that a customer announces each of its prefixes through each of its interfaces, producing an upper bound on the number of customer routes carried by the ISP. Taking the peering provider and customer routes together we arrive at the upper bound estimate of 5,511,277 routes for 214,417 prefixes or 25.7 routes per prefix on average for the month of June 2006.

For (b) ideally we would count the BGP updates received on eBGP sessions by each PE in the Tier-1 ISP network. Unfortunately we do not have access to this information, and therefore we make an estimate based on updates received by a route collector in the Tier-1's network. First we note that from the perspective of a router (or a route reflector) a destination has at most one best (most preferred) route at all times, though the identity of the best route may change over time. Such a best route corresponds to some external (CE-facing) network interface in the ISP that we call the 'best interface' for the destination (and again, whose identity changes over time). Since a router announces its best routes to the ISP's route collector, when we can examine the update rate received by the collector from the router for a given destination, we see the update rate corresponding to the best interface for the destination (for that router). Now making the simplifying assumption that each of the destination's external interfaces produces the same update rate, we have obtained an estimate of the update rate for any single interface for that destination based on that router. However, making this assumption causes some variance in the estimator across different routers, since different routers prefer different interfaces for a destination, and in reality these different interfaces do produce a different update rate. Therefore, we take the average of the estimates across all routers and route reflectors that feed the ISP's route collector, thereby reducing the variance of the estimator for per-destination, per-interface update rate. (In addition we note that we have ignored the effect of network-internal events such as IGP changes and route collector session resets. However these only make our estimator more conservative.) Now we use our overestimate on the number of routes per destination obtained under (a) previously and multiply that with our per-interface update rate to obtain the estimated upper bound on the total per-destination update rate observed by a deployed IRSCP server. Adding these estimates across all destinations gives the estimated upper bound on update rate to be processed by an IRSCP server. The route collector bins the updates received from its feeds by 15-minute intervals. We therefore base our estimates on the average update rates per time-bin. On the busiest day of the past year in May 2006 (busiest in terms of total updates received by the collector on a day), our estimator gives a maximum total update rate $rate_{all}$ of 4856 updates per second, and a 95th percentile and median value of 627 and 55, respectively.

We can also derive corresponding estimates for update rates sent by an IRSCP server to other IRSCP servers and to PEs and CEs. Given n_i deployed IRSCP servers each connected to a disjoint set of PEs and CEs (*e.g.*, in a deployment with one IRSCP server per POP n_i is just the number of POPs), on average each IRSCP server sends at a rate $rate_{all}/n_i$ to every other IRSCP server. Finally, an IRSCP server sends to each connected PE or CE the best route for each prefix. We estimate this rate $rate_{best}$ as $rate_{all}/avg.\#routes - per - prefix$ giving a maximum rate of 189, and 95th percentile and median values of 24 and 2.1 (on the busiest day).

4.5.4 Throughput

We determine the maximum value of input rate ($rate_{all}$) that the IRSCP server can sustain for an extended period of time, as follows. To discover the maximum input rate, we gradually increase the input rate and compare the observed output rate with an expected output rate of $rate_{all}/26 \cdot (n_{irscp} + n_{ibgp} + n_{ebgp})$. When the input rate is below its maximum, the output rate corresponds to the expected output rate. Once the input rate exceeds its maximum, the output rate steadily declines with increasing input rate.

Using this procedure we compare the performance of the standard BGP decision process with the explicitly ranked decision process, and evaluate the impact of the per-PE decision process. For the standard BGP decision process we find a maximum $rate_{all} \approx 3600$ updates/s, corresponding to an expected output rate of 40,846 updates/s. Figures 4.9(a) and (b) (*BGP DP, 15 PEs*) show the observed input and output rates during a run lasting twenty minutes, averaged over 30-second intervals. While the output rate is not as stable as the input rate, on average it corresponds to the expected output rate, and the figure shows that it is sustainable. Next, we load explicit rankings for 60,000 prefixes, each listing 26 egress IDs in some arbitrary order. In this case we find a maximum $rate_{all} \approx 2400$ updates/s, corresponding to an expected output rate of 27,231 updates/s. As expected, the explicitly ranked decision process is slower than the BGP decision process, since it runs in time quadratic rather than linear in the number of routes per prefix. Again, Figure 4.9 (*ranked DP, 15 PEs*) shows that the IRSCP server can sustain this workload.

Finally, to evaluate the impact of the per-PE decision process (vs. a single, router-wide decision process), we run an experiment again based on the BGP decision process, but in which all but one of the iBGP sessions between the IRSCP server and the update receiver are replaced by an eBGP session. In this case we find that the IRSCP server sustains a maximum $rate_{all} \approx 4200$ updates/s and produces the expected output rate of 47,654 updates/s (*BGP DP, 1 PE* in Figure 4.9). Figure 4.9(c) plots $rate_{all}$ for a varying number of PEs (and using the BGP decision process), but evaluated by sustaining the rate for only 40 seconds, rather than 20 minutes.

While the sustained throughputs are less than our maximum measurement-based estimate for $rate_{all}$ of 4900 updates/s, the IRSCP server easily manages to keep up with our 95th percentile estimate of 600 updates/s in all experiments, even when increasing the number of PEs to 50. Hence, we expect that an improved implementation of flow control in the IRSCP server should sufficiently slow down senders during times that the offered load exceeds the maximum sustainable throughput [40].

4.5.5 Memory consumption

We evaluate memory usage of an IRSCP server as a function of the number of routes it stores. We use a subset of the setup in Figure 4.10(c): the IRSCP server under test and the update generator. Also in this experiment we add BGP attributes from a routing table in the ISP network from October 2006. We vary the number of sessions between the update generator and the IRSCP server from 0 to 20 and measure the memory usage of the IRSCP server's RIB. We find a linear increase from 18.2 MB (0 sessions) to 226 MB.

Next, we examine memory usage of the explicit rankings. We configure an IRSCP server with rankings for 15 PEs and vary the number of ranked prefixes from 0 to 130,000 (but without loading their routes). Each ranking for a prefix and PE consists of 26 egress IDs. We measure the process size of `openbgpd`'s route decision engine, which stores the rankings. Again we find a linear increase from 840 KB (0 prefixes) to 994 MB (130,000 prefixes). Our current implementation is not optimized for space allocation, and, as a result cannot store rankings of this size for more than 130,000 prefixes. After we load 26 copies of the routing table, our prototype supports rankings for up to 75,000 prefixes on our test machines. However, we expect 26 egress IDs per prefix to be vastly more than needed in practice; five egress IDs per prefix can be stored for all 234,000 prefixes with full routing tables loaded.

4.6 Related work

In contrast with previous work on route control architectures [16, 29, 72], IRSCP implements a black box approach that gives an ISP full control over inter-domain routing in its network. To achieve this, we address the following challenges First, we give IRSCP *full visibility* of all routes available to the network through its eBGP interaction with neighboring networks. This is in contrast to an iBGP-only RCP [16], where routers in the IRSCP-enabled network only pass *selected* routes to IRSCP. Having full route visibility is needed to achieve full control (Section 4.3), and additionally prevents route oscillations within the network [12, 33, 49]. Second, we introduce a modified BGP decision process which we expose through a route control interface, allowing the ISP to control routing by means of a route control application. Third, we show how to distribute the IRSCP architecture, such that each replica is responsible for controlling a subset of

PEs. By contrast, RCP's server replication technique [16] requires each replica to scale to accommodate all PEs in the network. Similar to RCP, Bonaventure *et al.* [15] propose sophisticated route reflectors but limit their changes to the iBGP infrastructure.

The 4D project follows a clean slate approach that refactors the network architecture, creating a logically centralized control plane separate from forwarding elements [32, 80]. The IETF ForCES working group examines a separation of control plane and forwarding plane functions in IP network elements [43]. This work has a much narrower focus on defining the interface between control and forwarding elements, without considering interaction between different control plane elements. Once ForCES-enabled routers become available, IRSCP's iBGP communication with local routers might conceivably be replaced by a ForCES protocol. An earlier IETF proposal [23, 59] provides limited route control by defining a new BGP attribute subtype, the cost community, which can be assigned to routes and used to break ties at a certain 'point of insertion' in the BGP decision process. This proposal does not indicate under what conditions the cost community would be safe to use; by contrast, we show how our rankings should be constrained to ensure consistency.

4.7 Summary

In this chapter, we have presented a black box approach to the problem of route control in large ISPs, a feature needed by modern ISPs to support customer-oriented services, but lacking in the BGP protocol. More specifically, using a black box approach we have developed IRSCP, a server based architecture that permits an ISP to create an arbitrary assignment of available routes to its unmodified BGP routers. We now briefly review the main points of a black box approach with respect to our IRSCP architecture: the interface used by IRSCP to control routers and the extent to which the architecture satisfies the criteria for an effective black box approach.

We have chosen to use the BGP protocol itself as the external control interface to a BGP router — IRSCP sends BGP update messages to routers in order to announce, update, or withdraw routes. When sending a BGP update message for a route, IRSCP includes the route's BGP attributes (following today's practice) and tweaks the nexthop attribute to change the forwarding behavior of a router (Section 4.4.3). As such, IRSCP's control interface is clearly *portable*.

To understand whether IRSCP is *protocol-feasible*, we examine the protocols with which an IRSCP controlled router interacts with other routers. Of all the protocols potentially implemented by a router, the ones affected by IRSCP are BGP and IP forwarding. Since in the IRSCP architecture a BGP router no longer has BGP peering sessions with other routers, IRSCP is trivially protocol-feasible for BGP. While IRSCP changes the forwarding behavior of a router, it does so by sending standard BGP updates to the router, which must result in valid forwarding behavior in a standards compliant router. We have experimentally verified that this is the case in a testbed consisting of routers from multiple vendors.

Next, we turn to the question of *sufficient control*. There are two parts to this question. The first is whether IRSCP has sufficient control over BGP routers to implement packet forwarding as desired (*i.e.*, in accordance with routes selected by the explicitly ranked decision process). The second is whether IRSCP can send routing decisions to routers with adequate performance. To address the first part, we note that the means by which IRSCP configures a router's forwarding behavior — by sending a BGP update with the nexthop attribute carefully set to either a PE in the ISP's network or a CE in a neighboring network — is compliant with the BGP protocol standard. In the experiment mentioned above, we verify that routers from multiple vendors indeed forward packets exactly as specified by the attribute, for the various ways the attribute may be set. For the next part of the question of sufficient control, we observe that in order to correctly control routers in accordance with an ISP's policies, IRSCP not only needs to run a decision process for each router — it also needs to peer with a large number of routers in neighboring networks to acquire full visibility of available routes, as well as store and replicate these routes among multiple IRSCP servers. In our evaluation we have verified that, even under these constraints, an IRSCP implementation based on reasonable server hardware is adequately equipped to process and send updates to routers at a rate at least four times the 95th percentile of an ISP's routing load.

4.8 Acknowledgements

Chapter 4, in part, is a reprint of the material as it appears in Proceedings of the USENIX Annual Technical Conference, pages 295–308, Santa Clara, CA, June 2007. Verkaik, Patrick; Pei, Dan; Scholl, Tom; Shaikh, Aman; Snoeren, Alex C.; Van der Merwe, Jacobus. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Conclusions and future work

In this dissertation we have shown that a black box approach to enhancing networking protocols — one based on external control — is applicable to a wide range of challenging problems. We have covered the specific challenges of controlling at small time scale (Softspeak, Chapter 2), controlling adversarial devices (FairEst, Chapter 3), and scaling control to a large number of devices (IRSCP, Chapter 4). The decentralized nature of the underlying protocols we have considered (802.11 and BGP) poses an additional challenge for a black box approach, since the protocol may not offer a single point of control. As a result, the controllers for two of our solutions (Softspeak and IRSCP) are themselves decentralized.

Our approach in devising black box solutions to this range of problems has been to follow the three design criteria set out in Chapter 1, which we briefly summarize as follows. First, a black box solution should use an interface that offers *sufficient control* to modify the device's behavior in a manner that allows the black solution to meet its goals efficiently. Second, the control interface should be *portable*: a black box solution based on a non-portable interface has limited applicability. Third, the modified behavior of the controlled device should be *protocol-feasible*, *i.e.*, permitted by the protocols spoken by the device, so that the device may remain interoperable with other devices. These three criteria constrain the interactions of a controlled black box with its environment, in particular the controller and other devices, to maximize interoperability and controllability.

Ideally, a black box solution satisfies all three criteria perfectly. In practice, however, this may not be possible, and, given that each criterion imposes a constraint, a tradeoff between different criteria may be necessary. For example, to implement Softspeak's uplink TDMA scheme, we have used a control interface based on 802.11e features typically available in modern 802.11 hardware, rather than a strictly identical API — to our knowledge, a strictly identical API that provides sufficient control to implement TDMA does not exist. Thus, in this case, we trade off portability for sufficient control. As another example, a predecessor of Softspeak's unicast based downlink aggregation scheme used multicast to transmit aggregated VoIP packets [75]. Using multicast as a control interface may give better performance than Softspeak's unicast approach, which can suffer from overhearing problems, given that unicast packets are more likely to be sent at a less reliable MAC rate. In other words, the multicast interface scores better on the metric of sufficient control. However, in the presence of power saving clients, 802.11 access points send multicast packets at intervals that are a multiple of the beacon interval. Unfortunately, such an interval is too long for VoIP, making this solution protocol-infeasible for VoIP. If we were to make the solution protocol-feasible for VoIP, the access point would have to be modified to send multicast packets at an interval less than a beacon interval, which is protocol-infeasible for 802.11. Thus, in both cases, the solution compromises protocol-feasibility but improves sufficient control. Conversely, in using unicast packets to transmit aggregated packets, Softspeak trades off sufficient control for protocol-feasibility.

While we have shown that a black box approach may solve a variety of challenging problems, clearly it does not solve all manner of problems. We cannot expect to modify a networking device's behavior arbitrarily without changing its implementation. Furthermore, some problems might only be solvable by changing the specification of a networking protocol, *e.g.*, to convey information necessary to a solution. As an example, in Chapter 3, we have assumed that high bandwidth uploaders depend on a feedback loop through downlink traffic (similar to TCP). Once we drop this assumption, it may be the case that no black box solution exists that is able to enforce fairness while satisfying our three criteria. However, since we have devised the criteria as necessary conditions for a black box solution to be effective, in principle we should also be able to use them to identify the limits of applicability of a black box approach. Specifically, given a problem to be solved, if it can be shown that no controller exists that (a) has sufficient control over the networking device, (b) uses a portable interface to the device, and (c) allows the device to remain interoperable with other devices, then the problem cannot be solved using a black box approach. An interesting avenue for future work is to use formal modeling of protocols and devices to determine whether this is the case. In addition, such modeling might be used as a means to discover potential black box solutions.

In many cases, a black box approach may be one of the few options available to address emerging user needs for widely deployed networking protocols. Thus, while it is impossible for a protocol designer to anticipate every possible future user requirement, it is worth considering whether, instead, protocols can be designed and standardized in a way that simplifies the engineering needed to implement future black box solutions based on the protocol. In particular, we believe that a valuable addition to a protocol standard is a specification of (abstract) standard components of an implementation of the protocol, together with external interfaces to such components. The presence of such interfaces would improve control over devices in a portable manner, thus permitting a better tradeoff between sufficient control and portability. The 802.11 standard already partially follows such an approach — it defines an extensive set of functions that implementations are expected to support. Indeed, in controlling 802.11 hardware from software through programming interfaces, Softspeak makes use of a collection of 802.11e features that corresponds to such functions. In addition to programming interfaces, we believe that it may be useful to apply this principle to networked protocol interfaces, similar to OpenFlow [48]. As another example, BGP might define a protocol that allows a router's RIB to be exposed (similar to the MRT protocol). The presence of such an interface would have simplified the design of IRSCP, since IRSCP would not need to peer with routers in neighboring networks to learn the set of routes available to the ISP.

Ultimately, however, designers of networking protocols and devices cannot predict future user requirements, and so we should expect the gap between what protocols and devices offer, and what users need, to continue to exist. Until standards and technology are able to evolve at a much faster pace than they do today, we believe that a black box approach will remain essential in closing this gap.

Appendix A

IRSCP supplements

This appendix contains proofs for Section 4.3.2, as well as details of two aspects of the design of IRSCP: IGP distance changes and fault tolerance.

A.1 Proofs for claims in Section 4.3.2

Definition: *Deflection-free:* For each PE r: if egress route e is selected as best egress route for PE r, then some route f is selected as best egress route for PE $pe(e)$ (the PE incident on e) such that $pe(f) = pe(e)$.

In Figure 4.5, instantiating PE 2 for r and PE 3 for $pe(e)$ it should be obvious that Deflection-freedom prevents the anomalies shown in Figures 4.5(b) and (c). In Figure 4.5(a) PE 3 selects *some* route through itself (which is sufficient to prevent an anomaly), although we cannot tell from the figure if PE 3 has chosen e or some other route f for which $pe(f) = pe(e)$.

Claim: The BGP decision process in IRSCP is Deflection-Free.

Proof: Suppose for router r egress route e is selected as best egress route. If $r = pe(e)$ we are done, so assume $r \neq pe(e)$. Since e is in the egress set of r and all routers share the same egress set, e is also in the egress set of $pe(e)$. (Recall the definition of egress set in Table 4.1.) Note that for every route f: f is an eBGP-learned route from the perspective of $pe(e)$ iff $pe(f) = pe(e)$. Therefore the BGP decision process for $pe(e)$ has at least one eBGP-learned route when it enters Step B-5 and eliminates all non-eBGP-learned routes in Step B-5. Eventually it must select an eBGP-learned route. ■

Claim: If the explicit rankings given to an explicitly ranked decision process satisfy Ranking-Consistent-1 and Ranking-Consistent-2, then the explicitly ranked decision process is deflection-free.

Proof: Suppose for router r egress route e is selected as best egress route. We show that e is also selected as best egress route for router $pe(e)$. Since e is in the egress set of r and all routers share the same egress set, e is also in the egress set of $pe(e)$. We also know that e is most preferred among the routes in the egress set by Steps R-5 and R-6 of the explicitly ranked decision process for r, and that by Ranking-Consistent-1 the same egress links appear in r and $pe(e)$'s explicit rankings. There are two cases: e does or does not appear in the explicit rankings. If e does not appear in the explicit rankings, *none* of the routes in the egress set appear in the explicit rankings (or Step R-5 would have selected a different route for r). Therefore both $pe(e)$ and r identically rank the egress set using Step R-6, and $pe(e)$ selects e. If on the other hand e does appear in the explicit rankings, then r has selected it in Step R-5. Furthermore, with e available we know that $pe(e)$ must select *some* route e_2 that also appears in the explicit rankings (and in the egress set). Suppose $e_2 \neq e$. $e <_r e_2$ or Step R-5 would not have selected e for r. But from Ranking-Consistent-2 it follows that also $e <_{pe(e)} e_2$ and so $pe(e)$ cannot have selected e_2. ■

A.2 Handling IGP distance changes

We assume that the route control application has taken IGP distances into account when it creates the ranking. Although the IRSCP decision process could conceivably re-rank egress links in response to IGP distances, it generally does not do so for several reasons. First, for applications such as load balancing customer traffic, strict adherence to a shortest-path policy is of secondary importance. Indeed, tracking IGP distance changes can have adverse effects, such as causing large volumes of traffic to shift inadvertently [70]. The explicit ranking provided by an application introduces a degree of stability, effectively 'pinning' routes. If it is necessary to respond to IGP changes, we require the application to do so by providing an updated ranking. Teixeira *et al.* [69] suggest that in a large ISP with sufficient path diversity in its IGP topology the latency of MPLS tunnels is not greatly affected by IGP changes. For these cases, route pinning does not sacrifice much performance in terms of latency.

However, we do wish to handle the case in which IGP distances 'balloon' ex-

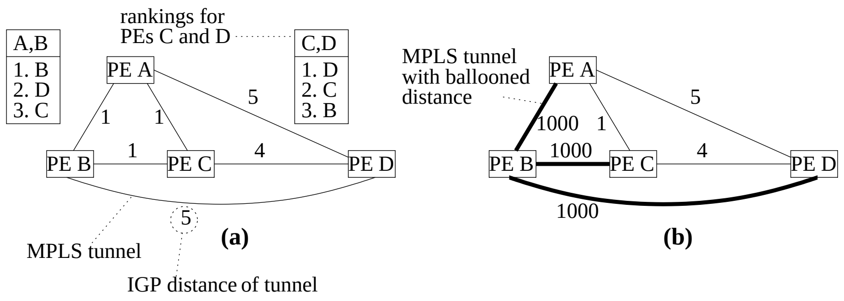


Figure A.1: Example of IGP ballooning. All routers are connected by MPLS tunnels whose distance metric is computed by IGP. (a) shows a topology based on which the application has computed a set of rankings. In (b) the distance of various tunnels has increased excessively.

cessively, effectively making some egress routes unusable. For example, this can occur when physical connectivity is disrupted and IGP diverts traffic around the disruption. Another example is router maintenance: typically the maintenance procedure involves setting the IGP distance between the router and the rest of the network to a very high value in order to gracefully move the traffic away from the router before it is brought down.

The network shown in Figure A.1 has three egress routes for some given destination: through PEs B, C and D. The application has assigned the egress route through PE B to PEs A and B and the egress route through D to PEs C and D. In Figure A.1(b) several IGP distances have ballooned, making PE A's preferred egress route through B virtually unusable for PE A, although A's rankings have not yet been updated.

We define an *Emergency Exit* procedure for cases such as this, which is as follows. If an IRSCP server finds that the IGP distance from a PE to the PE's preferred egress route balloons, the IRSCP server ignores the rankings for that PE and destination and reverts to hot-potato routing (*i.e.*, selects the nearest egress router, possibly the PE itself).[1] In the example, PE A overrides its ranking and chooses PE C. PE C's most preferred egress route (through D) has not ballooned and therefore PE C deflects to PE D, at which point the traffic egresses.

As this example shows, ignoring the rankings may lead to a deflection. We consider this acceptable, since (a) in a well-engineered network excessive ballooning

[1]However, if a black-hole egress ID is present in the ranking, then IRSCP still excludes egress routes that are ranked below the black-hole egress route before executing the hot-potato steps.

should be rare, and (b) at most one deflection (and no delayed blackholing) can occur, and therefore no forwarding loop can occur, as we prove below.

Claim: If PE R_1 forwards traffic to PE R_2, and if R_2 invokes Emergency Exit, then the traffic must egress the network at R_2 (without getting black-holed).

Proof: Since R_1 forwards traffic to R_2, it must have a route e in its egress set with $pe(e) = R_2$ and which is ranked higher than the black-hole egress ID b (independent of whether R_1 uses the explicitly ranked decision process or Emergency Exit). By the consistency constraints on the rankings and by the fact that all routers share the same egress set, R_2 has e in its egress set and ranks e above b. If R_2 now invokes Emergency Exit, it must select e (or some other local egress route other than b) in Step B-5 and egress the traffic. ■

From this it follows that the explicitly ranked decision process, enhanced with Emergency Exit, does not permit delayed black-holing.

Claim: Enhancing the explicitly ranked decision process with Emergency Exit permits at most one deflection in any forwarding path.

Proof: We consider traffic for some destination d that enters at some ingress PE A and assume to the contrary that two (or more) deflections do occur, *i.e.*, the traffic is forwarded from PE A to some PEs B, C and D, in that order. From the previous claim, we know that B and C do not invoke Emergency Exit, but instead follow the explicitly ranked decision process. Since B and C do not use themselves as the egress, it is clear that they could not have invoked Emergency Exit. Now consider a different (but concurrent) flow of traffic for d, ingressing at B. Since a router's forwarding behavior does not depend on where the traffic originates, this traffic follows the path $B - C - D$. In other words, this traffic is deflected without passing through a router that invokes Emergency Exit. This implies that the explicitly ranked decision process executed by B and C is not deflection-free, a contradiction. ■

A.3 Fault tolerance

We now discuss how IRSCP handles a number of common failure scenarios including the loss of customer connectivity to IRSCP, failure of individual IRSCP sessions and IGP failures, and route-control application communication failures.

First we examine failure in customer connectivity to IRSCP, by which we mean the ability for a CE to announce or learn reachability of a route to or from IRSCP.

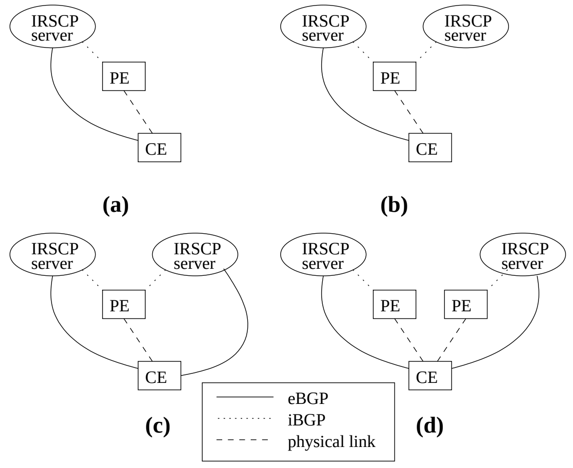


Figure A.2: Replicating customer connectivity. (a) No replication. (b) Replication of PE-IRSCP connectivity. (c) Replication of the CE-IRSCP eBGP session. (d) Replication of all components that make up customer connectivity.

As is apparent from Figure A.2(a), CE connectivity can be disrupted by failure of an IRSCP server, a PE, an eBGP session, an iBGP session or a physical link. (We consider failure of the CE the customer's responsibility.) Robustness to failure of any of these components can be improved by having the customer connect to several PEs and several IRSCP servers, as shown in Figure A.2(d). In addition, individual components can be made more robust as shown in Figure A.2(b) and (c).

Since IRSCP servers do not re-advertise updates learned from other IRSCP servers, each IRSCP server can only learn a particular route from one IRSCP server (*i.e.*, the IRSCP server that learned the route via an eBGP session with a router). A failure of an IRSCP session can therefore cause IRSCP servers to learn a different set of routes, potentially leading to inconsistency. Our aim is to prevent inconsistency from leading to a situation in which (a) IRSCP servers send updates to a PE that are inconsistent with updates sent to other PEs, (b) several IRSCP servers send inconsistent updates to the same PE. Among many potential causes underlying an IRSCP session failure we focus on misconfiguration of IRSCP peering and network partitioning.

We next discuss the case of a network (IGP) partitioning. By definition, a PE cannot have connectivity to IRSCP servers in different network partitions. Furthermore, there is no reachability between PEs in different network partitions and therefore no inconsistent routing between such PEs. It follows that it is sufficient to ensure consistency

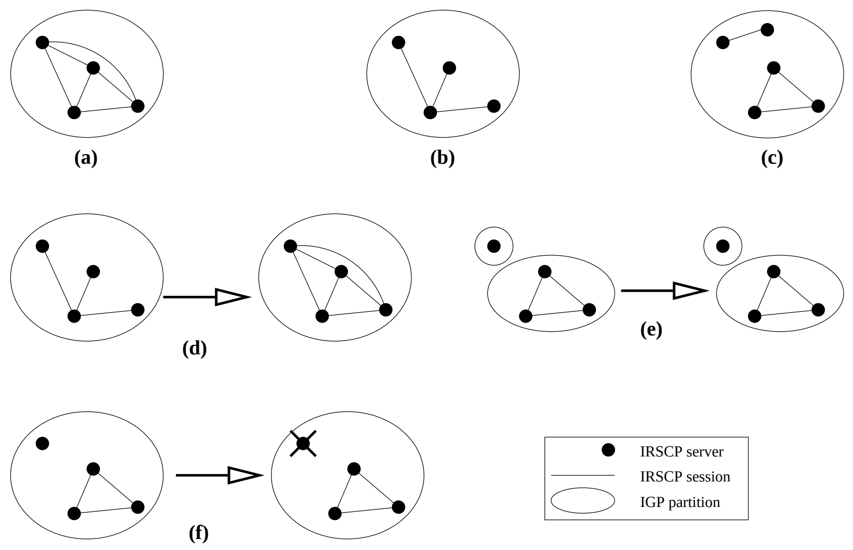


Figure A.3: Healing the IRSCP graph within a network partition. (a) Complete IRSCP graph. (b) Incomplete IRSCP graph. (c) Partitioned IRSCP graph. (d) Adding missing IRSCP peerings to an incomplete IRSCP graph. (e) A network partitioning. (f) Partitioned IRSCP graph in which all but one partition consists of a single IRSCP server.

among the IRSCP servers within each network partition separately.

Next we examine inconsistency within a network partition. We define an *IRSCP graph* consisting of IRSCP servers (vertices) and IRSCP sessions (edges) within a single network partition. In the absence of failure of IRSCP sessions within the network partition the IRSCP graph is complete and consists of a single connected component (Figure A.3(a)). Failures of IRSCP sessions within the IGP partition can cause the IRSCP graph to become incomplete (Figure A.3(b)) or even partitioned (Figure A.3(c)). We define procedures that 'align' each IRSCP graph partition with the corresponding network partition and 'heal' incomplete IRSCP graphs.

To handle failures within one partition in the IRSCP graph, we have IRSCP servers signal to each other (through the IRSCP sessions) which IRSCP servers are in their partition. If an IRSCP server I notices that it does not have an IRSCP peering with some IRSCP server J in its IRSCP partition, it establishes the peering with J (Figure A.3(d)). For failures that partition the IRSCP graph, we consider two sub cases. The case shown in Figure A.3(c) in which multiple IRSCP graph partitions contain more than one IRSCP server we assume to be highly unlikely, since it implies that all IRSCP servers in every such partition are misconfigured, and so we do not handle this case. If on the other hand an IRSCP server finds itself *alone* in an IRSCP graph partition (*i.e.*, it cannot establish any IRSCP sessions), it proceeds by checking its IGP viewer to see if any of the IRSCP servers it is trying to contact are present in its IGP partition. If none of them are, the IRSCP graph partition is in fact a network partition (Figure A.3(e)), and the IRSCP server continues to function correctly (see above). If the IRSCP server does find one of the other IRSCP servers in its network partition (Figure A.3(f)) it is highly likely that there is a problem with the IRSCP server itself. Rather than introducing potential inconsistencies, the IRSCP server halts. Here we rely on replication of IRSCP servers (discussed earlier) and let another IRSCP server take over.

Finally we look at failures related to the the route control application, specifically (a) omissions of PEs from the rankings generated by the application, and (b) failure of communication between the route control application and one or more IRSCP servers. When a PE is added to the network, the application must become aware of it so that the application can generate a corresponding per-PE ranking. However, relying on configuration is error-prone and may lead to persistent omissions of PEs from the rankings, potentially leading to inconsistency. Our solution is to rely on the Emergency Exit procedure (Section A.2): an IRSCP server that needs to execute the decision process for a PE for which it has not received ranking reverts to the BGP decision process. Similar to the Emergency Exit procedure this may lead to at most one deflection, and cannot lead to a forwarding loop.

When communication between the route control application and one or more IRSCP servers fails, some IRSCP servers may be excluded from ranking updates while others are not. A special case of this failure mode is that the route control application as a whole fails or is separated from IRSCP. With the rankings inconsistent the explicitly ranked decision process may no longer be deflection-free. To handle these failures, we proceed as follows. First, with each newly generated set of rankings (rankings concerning all PEs and all destinations), the application uploads the new ranking set to each IRSCP server, together with a version number that identifies the ranking set. (Of course, to reduce communication cost the application can transmit successive ranking sets using incremental updates.) Above, we described that IRSCP servers inform each other as to what IRSCP servers are present in their IRSCP graph partition. In this communication each IRSCP server ID is accompanied by the version number of the ranking set used by that IRSCP server. When an IRSCP server notices that it is no longer receiving rankings from the application it compares its version number with the version numbers of other IRSCP servers in the IRSCP graph partition. If it finds that some other IRSCP server has a more recent ranking set, it requests that ranking set from the other IRSCP server. As a result eventually all IRSCP servers in an IRSCP graph partition observe the same ranking set. Since as discussed each IRSCP graph partition is aligned with a network partition, it follows that eventually all IRSCP servers in the same network partition have the same ranking set.

A.4 Acknowledgements

Appendix A, in part, is a reprint of the material as it appears in Proceedings of the USENIX Annual Technical Conference, pages 295–308, Santa Clara, CA, June 2007. Verkaik, Patrick; Pei, Dan; Scholl, Tom; Shaikh, Aman; Snoeren, Alex C.; Van der Merwe, Jacobus. The dissertation author was the primary investigator and author of this paper.

Appendix B

FairEst supplements

B.1 Claims and proofs Section 3.3.1

Claim: If at time t, FairEst computes a rate limit based on client s, and $A_s(t) <= A_u(t)$ $(A_s(t) >= A_u(t))$ for all clients u, then $r(t) >= A_s(t)$ $(r(t) <= A_s(t))$.

Claim: FairEst eventually sets a's estimated demand as the lowest of all clients.

Proof: Let u be the client that currently has the lowest estimated demand, and therefore u is the client s_{min} based on which the current value of r was computed in the last round (Figure 3.8). By assumption (2), a uses less airtime than all others, and $A_a < r$. Thus, $a, u \in Free$, and FairEst updates their estimated demands, assigning a the lowest estimated demand among all $s \in Free$. However, suppose there are one or more clients that reach the rate limit and have a currently estimated demand lower than a's updated estimated demand. Let v be the client among them that has the lowest estimated demand. In that case, v determines the next rate limit and becomes a member of $Free$, at which point FairEst assigns it an estimated demand above a's. (Note that a remains a member of $Free$.) Ultimately, FairEst re-estimates the demand of all clients whose estimated demand is less than a's and eventually sets a's estimated demand as the lowest. ■

In the following claims, we assume that FairEst has succeeded at setting a's estimated demand as the lowest of all clients, and thus a is used as the s_{min} to compute rate limit r.

Claim: a's airtime usage either less than or greater than all other clients' airtime usage.

Proof: Suppose to the contrary that $A_u < A_a < A_v$. Therefore it must be the case that rate limit $r >= A_v$. However, in that case neither a nor u reach rate limit r, yet $A_u < A_a$, contradicting our assumption that a's airtime function is dominated by all others. ■

Next, suppose at time t, FairEst computes a rate limit $r(t)$ based on the set of airtime usage values $A_s(t)$. We examine FairEst's actions at time $t+1$. We define $A_s(t{+}1)'$, $A_{\neq s}(t{+}1)'$, $\ldots$, as FairEst's prediction (made at time t) of $A_s(t{+}1)$, $A_{\neq s}(t{+}1)$, $\ldots$.

Claim: Suppose a's airtime usage $A_a(t+1)$ is less than all other clients' airtime usage. FairEst will set $r(t+1)$ to a value $A_a(t+1) < r(t+1) < r(t)$.

Proof: From our assumption in the claim, $A_a(t{+}1) < r(t)$. Also, $A_{\neq a}(t{+}1) <= A_{\neq a}(t+1)'$, since $A_{\neq a}(t+1)$ is upper bounded due to the presence of rate limit $r(t)$. Since both $A_a(t+1)'$ and $A_{\neq a}(t+1)'$ were computed consistent with $E_a(t)$, and the E_a function is increasing in A_a and $A_{\neq a}$, it follows that $E_a(t{+}1) < E_a(t)$. Also, r's function is increasing in E_a, and so $r(t{+}1) < r(t)$. Finally, $r(t{+}1) >= A_a(t{+}1)$ from the earlier claim. ■

Claim: Suppose a's airtime usage $A_a(t+1)$ is greater than all other clients' airtime usage. FairEst will set $r(t+1)$ to a value $r(t) < r(t+1) <= A_a(t+1)$.

Proof: From Assumption (2), all clients other than a must be rate limited at $r(t)$, and thus (a) $A_a(t{+}1) > r(t)$, (b) $A_{\neq a}(t{+}1) = A_{\neq a}(t{+}1)'$. Similar to the reasoning of the previous claim, $r(t+1) > r(t)$ and $r(t+1) <= A_a(t+1)$. ■

Bibliography

[1] iCall for the iPhone (beta). http://www.icall.com/iphone/.

[2] Spectralink Inc - Spectralink Voice Priority (SVP). http://www.spectralink.com/files/literature/SVP_white_paper.pdf.

[3] Status of project IEEE 802.11e. http://tinyurl.com/2332nkf.

[4] Status of project IEEE 802.11n. http://tinyurl.com/28jr7.

[5] ITU-T P.800: Methods for subjective determination of transmission quality, August 1996.

[6] JiWire mobile audience insights report. http://www.jiwire.com/insights, Q1 2010.

[7] A. Akella, J. Pang, A. Shaikh, B. Maggs, and S. Seshan. A Comparison of Overlay Routing and Multihoming Route Control. In *ACM SIGCOMM*, September 2004.

[8] J. Al-Karaki and J. Chang. A simple distributed access control scheme for supporting QoS in IEEE 802.11 wireless LANs. *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, 1, 2004.

[9] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *ACM SOSP*, pages 131–145, October 2001.

[10] F. Anjum, M. Elaoud, D. Famolari, A. Ghosh, R. Vaidyanathan, A. Dutta, P. Agrawal, T. Kodama, and Y. Katsube. Voice performance in WLAN networks-an experimental study. *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, 6, 2003.

[11] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and control in gray-box systems. *SIGOPS Oper. Syst. Rev.*, 35(5):43–56, 2001.

[12] A. Basu, C.-H. L. Ong, A. Rasala, F. B. Shepherd, and G. Wilfong. Route Oscillations in I-BGP with Route Reflection. In *ACM SIGCOMM*, pages 235–247, 2002.

[13] T. Bates, R. Chandra, and E. Chen. BGP Route Reflection - An Alternative to Full Mesh IBGP. RFC 2796, April 2000.

[14] N. Blefari-Melazzi, A. Detti, I. Habib, A. Ordine, and S. Salsano. Tcp fairness issues in ieee 802.11 networks: Problem analysis and solutions based on rate control. *Wireless Communications, IEEE Transactions on*, 6(4):1346–1355, April 2007.

[15] O. Bonaventure, S. Uhlig, and B. Quoitin. The case for more versatile BGP Route Reflectors. Internet draft: draft-bonaventure-bgp-route-reflectors-00.txt, July 2004.

[16] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a Routing Control Platform. In *ACM/USENIX NSDI*, 2005.

[17] M. Caesar and J. Rexford. BGP routing policies in ISP networks. IEEE Network, November 2005.

[18] K. Cai, M. Blackstock, R. Lotun, M. J. Feeley, C. Krasic, and J. Wang. Wireless unfairness: alleviate mac congestion first! In *WinTECH '07: Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 43–50, New York, NY, USA, 2007. ACM.

[19] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying Skype User Satisfaction. ACM SIGCOMM, September 2006.

[20] K.-T. Chen, P. Huang, G.-S. Wang, C.-Y. Huang, and C.-L. Lei. On the Sensitivity of Online Game Playing Time to Network QoS. IEEE Infocom, April 2006.

[21] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benkö, J. Chiang, A. C. Snoeren, S. Savage, and G. M. Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *Proceedings of the ACM SIGCOMM Conference*, Kyoto, Japan, August 2007.

[22] S. Choi, K. Park, and C.-k. Kim. Performance impact of interlayer dependence in infrastructure wlans. *IEEE Transactions on Mobile Computing*, 5(7):829–845, 2006.

[23] Cisco Systems. BGP Cost Community. Cisco IOS Documentation.

[24] R. G. Cole and J. H. Rosenbluth. Voice over ip performance monitoring. *SIGCOMM Comput. Commun. Rev.*, 31(2):9–24, 2001.

[25] L. Ding and R. Goubran. Speech quality prediction in voip using the extended e-model. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 7:3974–3978 vol.7, Dec. 2003.

[26] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. Sicker, and D. Grunwald. MultiMAC-An Adaptive MAC Framework for Dynamic Radio Networking. *IEEE DySPAN*, 2005.

[27] N. Duffield, K. Gopalan, M. R. Hines, A. Shaikh, and J. E. van der Merwe. Measurement Informed Route Selection. Passive and Active Measurement Conference, April 2007. Extended abstract.

[28] J. Farrell and G. Saloner. Coordination through committees and markets. *The RAND Journal of Economics*, 19(2):235–252, Summer 1988.

[29] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. E. van der Merwe. The Case for Separating Routing from Routers. ACM SIGCOMM FDNA, Aug 2004.

[30] N. Feamster and J. Rexford. A Model of BGP Routing for Network Engineering. In *SIGMETRICS*, June 2004.

[31] S. Garg and M. Kappes. An experimental study of throughput for udp and voip traffic in ieee 802.11b networks. *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, 3:1748–1753 vol.3, 16-20 March 2003.

[32] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM CCR*, 35(5), 2005.

[33] T. Griffin and G. T. Wilfong. Analysis of the MED Oscillation Problem in BGP. In *ICNP*, 2002.

[34] T. G. Griffin and G. Wilfong. On the Correctness of IBGP Configuration. In *ACM SIGCOMM*, pages 17–29, New York, NY, USA, 2002. ACM Press.

[35] F. Guo and T. Chiueh. Software TDMA for VoIP applications over IEEE802.11 wireless LAN. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2366–2370, May 2007.

[36] J. Ha and C.-H. Choi. Tcp fairness for uplink and downlink flows in wlans. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–5, 27 2006-Dec. 1 2006.

[37] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[38] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 836–843 vol.2, March-3 April 2003.

[39] G. Huston. Wither Routing? The ISP Column, November 2006.

[40] G. Huston. Damping BGP. The ISP Column, June 2007.

[41] J. Karlin, S. Forrest, and J. Rexford. Pretty Good BGP: Improving BGP by Cautiously Adopting Routes. In *ICNP*, November 2006.

[42] T. Kawata, S. Shin, A. Forte, and H. Schulzrinne. Using Dynamic PCF to Improve the Capacity for VoIP Traffic in IEEE 802.11 Networks. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, 2005.

[43] H. Khosravi and T. Anderson. Requirements for Separation of IP Control and Forwarding. IETF RFC 3654, November 2003.

[44] S. Kim, B. Kim, and Y. Fang. Downlink and Uplink Resource Allocation in IEEE 802.11 Wireless LANs. *Vehicular Technology, IEEE Transactions on*, 54(1):320–327, 2005.

[45] S. W. Kim, B.-S. Kim, and Y. Fang. Downlink and uplink resource allocation in ieee 802.11 wireless lans. *Vehicular Technology, IEEE Transactions on*, 54(1):320–327, Jan. 2005.

[46] R. R. Kompella, S. Ramabhadran, I. Ramani, and A. C. Snoeren. Cooperative packet scheduling via pipelining in 802.11 wireless networks. In *Proceedings of the Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, pages 35–40, Philadelphia, PA, August 2005.

[47] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. *SIGCOMM Comput. Commun. Rev.*, 32(4):3–16, 2002.

[48] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[49] D. McPherson, V. Gill, D. Walton, and A. Retana. Border Gateway Protocol (BGP) Persistent Route Oscillation Condition. RFC 3345, August 2002.

[50] K. Medepalli, P. Gopalakrishnan, D. Famolari, and T. Kodama. Voice capacity of IEEE 802.11b, 802.11a and 802.11g wireless LANs. In *Proc. IEEE Global Telecommunications Conference*, 2004.

[51] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM Conference*, Barcelona, Spain, August 2009.

[52] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMACflexible wireless research platform. *Proc. HotNets-IV.*

[53] J. Nichols and M. Claypool. The Effects of Latency on Online Madden NFL Football. In *ACM NOSSDAV*, June 2004.

[54] G. Paschos, I. Papapanagiotou, S. Kotsopoulos, and G. Karagiannidis. A New MAC Protocol with Pseudo-TDMA Behavior for Supporting Quality of Service in 802.11 Wireless LANs. *EURASIP Journal on Wireless Communications and Networking*, 6:76–84, 2006.

[55] N. Patrick, T. Scholl, A. Shaikh, and R. Steenbergen. Peering Dragnet: Anti-Social Behavior Amongst Peers, and What You can Do About It. NANOG 38, October 2006.

[56] S. Pilosof, R. Ramjee, D. Raz, Y. Shavitt, and P. Sinha. Understanding tcp fairness over wireless lan. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 863–872 vol.2, March-3 April 2003.

[57] A. Rao and I. Stoica. An overlay MAC layer for 802.11 networks. *Proc. of Mobile Systems, Applications and Services (Mobisys '05)*, 2005.

[58] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). IETF RFC 4271, 2006.

[59] A. Retana and R. White. BGP Custom Decision Process. Internet draft: draft-retana-bgp-custom-decision-00.txt, October 2002.

[60] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-End Effects of Internet Path Selection. In *ACM SIGCOMM*, September 1999.

[61] A. Sharma, M. Tiwari, and H. Zheng. MadMAC: Building a reconfigurable radio testbed using commodity 802.11 hardware. *Proc. First IEEE Workshop on Networking Technologies for Software Defined Radio (IEEE SECON 2006 Workshop).*

[62] S. Shin and H. Schulzrinne. Balancing uplink and downlink delay of VoIP traffic in WLANs using Adaptive Priority Control (APC). In *International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*, 2006.

[63] S. Shin and H. Schulzrinne. Experimental Measurement of the Capacity for VoIP Traffic in IEEE 802.11 WLANs. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2018–2026, 2007.

[64] T. Simcoe. Delays and de jure standards: What caused the slowdown in internet standards development?, 2004.

[65] J. Snow, W. Feng, and W. Feng. Implementing a low power TDMA protocol over 802.11. *Wireless Communications and Networking Conference, 2005 IEEE*, 1, 2005.

[66] G. Tan and J. Guttag. Long-term time-share guarantees are necessary for wireless lans. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 35, New York, NY, USA, 2004. ACM.

[67] G. Tan and J. Guttag. Time-based fairness improves performance in multi-rate wlans. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 23–23, Berkeley, CA, USA, 2004. USENIX Association.

[68] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings of the IEEE Infocom Conference*, Barcelona, Spain, April 2006.

[69] R. Teixeira, T. G. Griffin, M. G. C. Resende, and J. Rexford. TIE Breaking: Tunable Interdomain Egress Selection. In *CoNEXT*, 2005.

[70] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of Hot-Potato Routing in IP Networks. In *ACM SIGMETRICS*, 2004.

[71] N. H. Vaidya, P. Bahl, and S. Gupta. Distributed fair scheduling in a wireless lan. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 167–178, New York, NY, USA, 2000. ACM.

[72] J. E. van der Merwe et al. Dynamic Connectivity Management with an Intelligent Route Service Control Point. ACM SIGCOMM INM, October 2006.

[73] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, CA, February 2009.

[74] P. Wang, H. Jiang, and W. Zhuang. Capacity Improvement and Analysis for Voice/Data Traffic over WLAN. *IEEE Transactions on Wireless Communications*.

[75] W. Wang, S. C. Liew, and V. Li. Solutions to performance problems in voip over a 802.11 wireless lan. *Vehicular Technology, IEEE Transactions on*, 54(1):366–384, Jan. 2005.

[76] W. Wang, S. C. Liew, Q. Pang, and V. O. K. Li. A multiplex-multicast scheme that improves system capacity of voice-over-ip on wireless lan by 100In *ISCC '04: Proceedings of the Ninth International Symposium on Computers and Communications 2004 Volume 2 (ISCC"04)*, pages 472–477, Washington, DC, USA, 2004. IEEE Computer Society.

[77] J. Wu, Z. Mao, J. Rexford, and J. Wang. Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network. In *USENIX NSDI*, 2005.

[78] Y. Wu, Z. Niu, and J. Zhu. Upstream/downstream unfairness issue of tcp over wireless lans with per-flow queueing. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 5, pages 3543–3547 Vol. 5, May 2005.

[79] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast, long distance networks. In *Proceedings of the IEEE Infocom Conference*, Hong Kong, China, March 2004.

[80] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4D Network Control Plane. In *ACM/USENIX NSDI*, 2007.

[81] J. Yu, S. Choi, and J. Lee. Enhancement of VoIP over IEEE 802.11 WLAN via Dual Queue Strategy. *Proc. IEEE ICC*, 4, 2004.

[82] L. Zhang. A retrospective view of NAT. *IETF Journal*, 3(2), October 2007.