UNIVERSITY OF CALIFORNIA
RIVERSIDE

Meeting the Challenges of Software-Based Networks and Services

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

Doctor of Philosophy

in

Computer Science

by

Ali Mohammadkhan

September 2019

Dissertation Committee:

Prof. K.K. Ramakrishnan, Chairperson
Prof. Nael Abu-Ghazaleh
Prof. Jiasi Chen
Prof. Srikanth V. Krishnamurthy

The Dissertation of Ali Mohammadkhan is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to use this opportunity to thank everyone who has helped and supported me during my Ph.D..

First, I would like to thank my advisor, Prof. K. K. Ramakrishnan who had a crucial role in directing and steering me during my studies. In addition, I would like to thank the other Professors of my committee who helped me with their invaluable comments. Prof. Jiasi chen, Prof. Nael Abughazaleh, Prof. Srikanth Krishnamurthy, Prof. Vassilis J. Tsotras, Porf. Laxmi N. Bhuyan, and Prof. Nanpeng Yu, thank you for accepting to be a part of my committee during different stages of my Ph.D..

I would like to thank my wife, Sheida Ghapani, not only because of her unconditional support in my personal life, but also because of the time she spent to teach me about optimization and helped me to learn how to formulate problems for optimization.

I am grateful for all the friends I had in our group and in our lab, they wholeheartedly helped me during these years. Space will not be enough to list them all but at least I would like to name a few: Mohammad Jahanian, Shahryar Afzal, Sourav Panda, Aditya Dhakal, and Sameer G. Kulkarni, thank you all.

I would like to thank our collaborator that I was lucky to experience the collaboration with them. From George Washington University, Prof. Timothy Wood, Wei Zhang, and Grace Liu. From Norwegian University of Science and Technology, Y.T. Woldeyohannes and Prof. Yuming Jiang. From Future Networks, Huawei Technologies, Uma Chunduri, and Kiran Makhijani.

The text of this dissertation, in full (unless explained otherwise in the detailed description of each publication below), is a reprint of the material as it appears in the following publications. The co-author, Prof. K. K. Ramakrishnan, listed in all the publications, directed and supervised the research which forms the basis for this thesis. For the sake of brevity, I mentioned his key role here and I will not repeat this sentence for all the publications one by one.

- "CleanG - Improving the Architecture and Protocols for Future Cellular Networks with NFV, submitted to ACM/IEEE Transaction on Networking Journal, 2019. I was the main author of this work and this work was done under Prof. Ramakrishnan's supervision.

- "Re-Architecting the Packet Core and Control Plane for Future Cellular Networks, The 27th IEEE International Conference on Network Protocols, 2019. I was the main author of this work and this work was done under Prof. Ramakrishnan's supervision.

- "P4NFV: P4 Enabled NFV Systems with SmartNICs, in Proc. of IEEE Conference on Network Function Virtualization and Software Defined Networks, 2019, beside I as the main author and Prof. Ramakrishnan, we had the opportunity to work with Sameer G. Kulkarni and Sourav Panda. Sameer G. Kulkarni helped us by providing his technical expertise for shaping a better paper and Sourav Panda contributed specially in SmartNIC component of the architecture and its evaluation.

- "Re-Architecting the Packet Core and Control Plane for Future Cellular Networks," in Proc. of 27th IEEE International Conference on Network Protocols (ICNP), 2019. Authors of this work were Prof. Ramakrishnan and I.

- "Improving Performance and Scalability of Next Generation Cellular Networks," in IEEE Internet Computing, vol. 23, 2019. Beside I as the main author and Prof. Ramakrishnan, Uma Chunduri and Kiran Makhijani from Huawei Technologies mostly contributed to the transportation protocol section and improving the paper based on their technical expertise. The transportation protocol part is removed from this thesis.

- "A scalable resource allocation scheme for NFV: Balancing utilization and path stretch," Y. T. Woldeyohannes is the main author of this work. Mostly, I helped with overall problem formulation and few suggestions for example in the heuristic part to improve the perform ace of the suggested approach.

- "ClusPR: Balancing Multiple Objectives at Scale for NFV Resource Allocation,". Y. T. Woldeyohannes is the main author of this work. Similar to the previous work, mostly, I helped with overall problem formulation and few suggestions, for example in the heuristic part to improve the performance of the suggested approach. In addition, I helped with evaluation part as well.

- "CleanG: A Clean-Slate EPC Architecture and Control Plane Protocol for Next Generation Cellular Networks". I was the main author and it was done under Prof. Ramakrishnan's supervision. Ashok Sunder Rajan, and Christian Maciocco from Intel helped with estimating the load for the next generation of cellular networks.

- "SDNFV: Flexible and Dynamic Software Defined Control of an Application- and Flow-Aware Data Plan". Wei Zhang is the main author of this work. I mostly helped with the placement challenges in this framework.

- Considerations for Re-Designing the Cellular Infrastructure Exploiting Software-Based Networks. I was the main author of this work and this work was done under Prof. Ramakrishnan's supervision. Ashok Sunder Rajan, and Christian Maciocco from Intel helped with their estimating expected load in next generation of cellular network base on their experience in the cellular network field.

- Protocols to Support Autonomy and Control for NFV in Software Defined Networks. I was the main author of this work and this work was done under Prof. Ramakrishnan's supervision. We used the expertise of other authors in the software defined networking and network function virtualization challenges.

- Virtual Function Placement and Traffic Steering in Flexible and Dynamic Software Defined Networks. I was the main author of this work and this work was done under Prof. Ramakrishnan's supervision. We used the expertise of other authors in the software defined networking and network function virtualization challenges.

  My work was partially supported by following grants:

  - NSF grants CNS-1522546, CRI-1823270, and CNS-1763929.
  - ARO DURIP grant W911NF-15-1-0508.
  - and a grant from Futurewei Inc. (Huawei Tech. Co. Ltd.s HIRP Grant)

This thesis is dedicated to my wife, my parents, and my sister

who have always supported and encouraged me.

ABSTRACT OF THE DISSERTATION

Meeting the Challenges of Software-Based Networks and Services

by

Ali Mohammadkhan

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2019
Prof. K.K. Ramakrishnan, Chairperson

Nowadays, it is possible to have high-performance software-based network functions. This concept, known as Network Function Virtualization (NFV), enables us to run network functions on-demand and where they are needed. Another aspect of network softwarization is Software-defined networking (SDN), which separates data and control plane and a logically centralized controller controls data plane. The computer network enabled by SDN and NFV has unique and interesting challenges and opportunities. In these networks, network functions can be instantiated all over the network, and the flows are steered through them, which is known as service chaining. In one branch of our work, we showed how jointly solving routing, and network function placement problem outperforms traditional placement solutions. Next step was designing a protocol for service chaining in these networks. Hence, we showed that efficient use of available information in the centralized controller makes the protocol more efficient with a reduces the number of messages and bits in the headers. Thus far, we had considered nodes as black boxes, but in the next branch of our work, we focused on each node. We proposed a solution for the architec-

ture of a protocol-free software switch equipped with SmartNICs and the optimization of resources to carry out different tasks within each node. Based on the lessons we learned in the projects above, we worked on the application of these technologies in the cellular domain. We proposed an NFV-based architecture and protocol for the cellular packet core. Our proposed architecture, CleanG, is simple, scalable, and efficient. In addition, in the CleanG protocol, the number of control messages exchanged is reduced dramatically, and packets are forwarded through more efficient tunneling. This reduction in messages lowers the delay and the load on control plane components, which increases the system capacity dramatically. In conclusion, software-based networks provide a plethora of opportunities for the next generations of networks. However, to leverage them efficiently, we believe merely implementing hardware components as a software piece is not the answer. Thus, it is crucial to rethink the architecture and protocols, and the specific challenges and opportunities of software-based networks should be considered.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Softwarization of computer networks is affecting different aspects of these networks and two of its main manifestations are software-defined networking and network function virtualization.

Software-Defined Networking (SDN) provides a logically centralized control plane for network service providers, enabling them to program and control the network forwarding plane (for example, SDN controller like ONOS [27]). Software-defined networking promises to provide greater flexibility for precisely directing packet flows using a software-based control plane for the network. The underlying assumption is that the data plane is simple, comprising commodity switches, with little or no state other than the forwarding table of 'match-action' rules populated by a logically centralized SDN controller. Software-defined networking introduces the concept of separation between the data plane and control plane, to provide more flexibility in how individual flows are handled [74, 7].

However, today's networks are much more than just packet forwarding entities, with complex network services prevalent in custom-built middlebox systems at the edges of the network. Network Function Virtualization (NFV) has emerged as a technique to run high-performance network services as software running in virtual machines (VMs) on commodity servers. Improved techniques for packet processing in virtualized platforms running on commercial off the shelf (COTS) systems make it possible to run network functions on software-based platforms rather than purpose-built hardware appliances [108, 98, 142]. NFV thus enables easy deployment of software-based network functions dynamically in the network, at a much lower cost. Both these technologies promise to radically alter how networks are deployed and managed, offering greater flexibility and enabling network services to be added on demand.

One of the primary outcomes of SDN and NFV is an effortless service chaining. Packet flows of different may need to go through a different sequence of services. For example, a flow may need to be checked by the firewall, and intrusion detection system and later be compressed by another network function (NF). The procedure of running different sequence of NFs on different flows is called service chaining. By using SDN, it is possible to have centralized control over the flow path, and by using NFV, it is possible to instantiate different Network Functions when and where it is necessary. Hence, these two techniques provide the opportunity for an easy and configurable service chaining.

As we mentioned, SDN and NFV provide flexibility and dynamic capability in the control and data planes and make it possible to have service chaining for different flows. Service chaining is crucially depending on dynamic instantiation and placement of network

functions (NF) in the network and flexible routing of the flows through them. In Chapter 2, we proposed a mixed integer linear formulation of the problem, and we proposed heuristics for a faster solution to the problem.

While in Chapter 2, we worked on the placement problem to place different functions on different processing nodes, in Chapter 3, we studied the optimization of placement within a node. To improve the performance and to free CPU resources on a software switch, it is possible to assign some of the tasks to the SmartNIC and handle the rest by the main host. In this chapter, we propose a new architecture for software-based protocol-independent switches and NF hosts. In this architecture, a major component, called SDN Agent, abstract the node as a single entity to the SDN controller. However, this node can be made of a software switch and a SmartNIC. This component breaks the P4 component to the tables and actions that can be carried out by the software host or the SmartNIC. This problem is complex as it depends on lots of different constraints, such as PCI Express bandwidth, delay, and data dependency. We formulated this problem as another mixed integer linear problem and tried to minimize delay the processing within each node.

After deciding for the routing of the flows and placing the NFs, and optimal handling of the flow within a node, a protocol is needed. The role of this protocol is to specify the set of NFs for each flow, recording the progress of the flows in the service chain. For example, a flow may visit a processing node twice for receiving two different services. We need to differentiate the first and the second visit to provide different services each time. in Chapter 4 we investigate this problem in depth.

In the first three chapters, we mentioned the challenges and opportunities of software-based networks. In the rest of the thesis, we focus on the application of the techniques that we mentioned on the cellular network domain. Cellular networks have evolved from just providing voice communication between people to ubiquitous data, video, and voice connectivity for people as well as supporting machine-to-machine (M2M) and Internet of Things (IoT) communication. However, cellular networks continue to face significant capacity, latency, and scalability challenges. While some of these concerns are being addressed in the 5G networks currently being deployed, fundamental problems remain, and the current solution that involve implementing each traditional component as a separate Network Functions is not efficient enough and more efficient use of Network Function Virtualization is necessary. We believe it is highly desirable to take a new look at the architecture and associated protocols with the goal of improving performance to meet upcoming challenges. In Chapter, 5, we investigate the requirement and necessary performance needs in the next generations of the cellular network. In the next chapter, 6, we explained the motivation and initial design of protocol and architecture. In Chapter 7, we completed the picture with the in-depth explanation and details of the proposed architecture and protocol for the next generation of cellular networks. Before concluding this dissertation, in Chapter 8 we have briefly covered other works that are related to the subject of this thesis that I was a co-author in them but I was not the main author unlike the other chapters.

Chapter 2

Virtual function placement and traffic steering in flexible and dynamic software defined networks

The integration of network function virtualization (NFV) and software defined networks (SDN) seeks to create a more flexible and dynamic software-based network environment. The line between entities involved in forwarding and those involved in more complex middle box functionality in the network is blurred by the use of high-performance virtualized platforms capable of performing these functions. A key problem is how and where network functions should be placed in the network and how traffic is routed through them. An efficient placement and appropriate routing increases system capacity while also minimizing the delay seen by flows.

In this chapter, we formulate the problem of network function placement and routing as a mixed integer linear programming problem. This formulation not only determines the placement of services and routing of the flows, but also seeks to minimize the resource utilization. We develop heuristicsto solve the problem incrementally, allowing us to support a large number of flows and to solve the problem for incoming flows without impacting existing flows.

2.1 Introduction

A service provider network rarely consists of just forwarding entities like switches and routers [201]. Current networks commonly include middlebox functions such as firewalls, proxies, caches, policy engines, etc. Switches and middlebox functionality can also coexist on the same COTS platform with the use of NFV. Flows have to be routed through these network functions in a pre-defined order, and SDN provides the necessary power and flexibility to achieve this. A network function (NF) can be dynamically instantiated in a host as long as there is enough computational power for hosting the service. Flows steered through switches and NFs, with the goal of executing the needed service functions in the required order. This could potentially result in the flow having to traverse a given link multiple times (i.e. even having loops as perceived by the network layer, [15]). Thus, the placement of the NFs and steering flows through them need to be done judiciously. The focus of this paper is on placing the NFs and routing of flows, ensuring that each flow's path starts from an entry switch, meets all the necessary NFs in sequence, and ends at the exit switch.

Multiple studies on middlebox or virtual machine (VM) placement consider the placement problem independent from how flows utiize these functions and the routing of flows. Some, such as [243, 81] consider both placement and steering of flows, but solve them separately. For instance, a heuristic is used for placement and use its result as an input for flow steering. However, a placement solution that does not leverage the information about the flows can be inefficient.

In this paper we have formulated the service placement and flow steering problems jointly in a single mixed integer linear problem (MILP) formulation. This formulation results in the optimal placement of services and the routing of the flows. It seeks to minimize the maximum link and CPU core utilization and the maximum delay of flows in the network. This approach also offers the opportunity to solve problems incrementally, as flows are added. This means that instead of solving a large problem which may be intractable, the problem is partitioned and solved in smaller pieces, and the final result is still close to the optimal solution. Another benefit of the incremental solution is that after adding new flows to the network, we only have to solve the partial problem of newly added flows without impacting existing flows.

2.2 System Description

Network nodes in our software-based network play multiple roles – providing the conventional role of forwarding packets, and supporting network services such as firewalls, proxies, policy engines etc.. A COTS system CPU comprising multiple cores could be assigned to forwarding or to provide network functions. To avoid the overheads of Non-

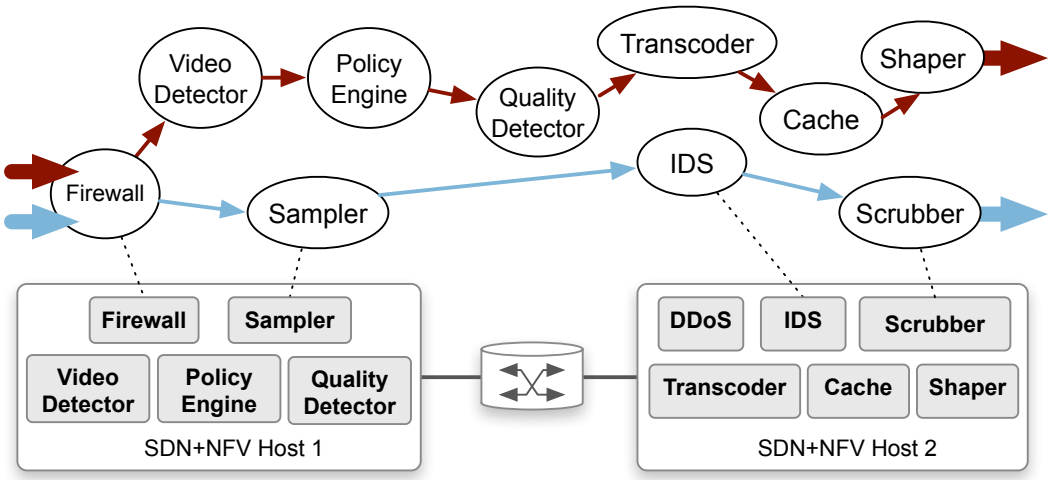


Figure 2.1: NF Service Graph Map across Multiple Hosts

uniform memory access (NUMA), we assume that a core is dedicated to a single VM that supports a network service or forwarding function.

The desired functions are instantiated in the network based on the requirements of each flow and the placement decisions are made based on the estimated per-packet computation requirement for the function and the link bandwidth as well as the maximum tolerable delay for the flow. In addition, the set (or chain) of network services and their order is the same for all packets of the flow.

Two examples of the services chains are depicted in Figure 2.1. Each service has its computational requirements, so we set a limit on maximum number of flows a service can support on a specified hardware. Making placement decision for services need to take all of these factors into consideration. For example, in Figrue 2.2 we have a network with 8 switches (we will henceforth use the term switches and network nodes interchangeably).

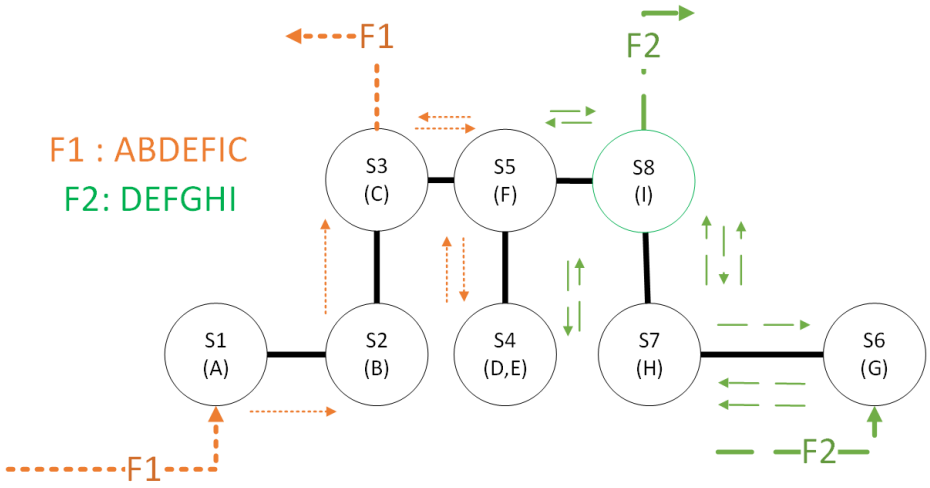


Figure 2.2: An example of service placement for two flows

All the switches except $S4$ have just one free core, while switch $S4$ has two available cores. The assigned services at each switch is shown in parentheses. The service chain for $F1$ is "ABDEFIC" and the service chain for $F2$ is "DEFGHI" (each letter represent a service like a firewall or a proxy.) The difficulty for efficient placement increases dramatically with the growth in the number of flows or network nodes.

2.3 MILP Formulation

A goal of the formulation is to obtain an 'efficient' placement of services and routing of the flows without violating the constraints of the maximum capacity of the links and tolerable delays of flows. The 'efficient' placement seeks to minimize the utilization of the links and of the available CPU cores, thus maximizing system capacity. This is especially important when we need to solve the problem incrementally as new flows and

functionality may be dynamically added to the network. Our proposed problem formulation is as follows:

Minimize U subject to:

$$\forall k \in Flows, \forall i, m \in Switches, \forall j \in Services, \forall l \in O_k,$$

$$\forall l' \in O'_k :$$

$$\sum_j M_{ij} \leq C_i \tag{2.1}$$

$$X_{K_{il}} = M_{ij} S_{k_{jl}} \tag{2.2}$$

$$N_{k_{il}} = X_{K_{il}} \circ W_{k_{il}} \tag{2.3}$$

$$\sum_i N_{k_{il}} = \sum_j S_{k_{jl}} \tag{2.4}$$

$$F_k = [I_k N_k E_k] \tag{2.5}$$

$$\sum_m V_{k_{l'_{im}}} - \sum_m V_{k_{l'_{mi}}} = F_{k_{il'}} - F_{k_{i(l'+1)}} \tag{2.6}$$

$$\sum_{l'} \sum_{i,m} V_{k_{l'_{im}}} D_{im} \leq T_k \tag{2.7}$$

$$A_k = S_k N_k^T \tag{2.8}$$

$$\sum_k A_{k_{ji}} \leq P_{ji} \tag{2.9}$$

$$\sum_k \sum_{l'} (V_{k_{l'_{im}}} \circ B_k) \circ (1/H_{im}) \leq U \tag{2.10}$$

$$\sum_k A_{k_{ji}} / P_{ji} \leq U \tag{2.11}$$

The definition of variables in this formulation is provided in Table 2.1. After obtaining the solution, the placement result is stored in variable M and the routing steps

Var.	Definition
U	Maximum utilization of links and switches
M_{ij}	Number of running instances of service j on switch i
C_i	Number of available cores on switch i
$N_{k_{il}}$	Selected switch for order l of the flow k's service chain
$S_{k_{ij}}$	This value is one, if i is the j^{th} service in flow k's service chain; zero otherwise
W_k	Binary decision variable for satisfying constraint (2.4)
I_{ki}	Equal to 1, if i is the entrance switch for flow k
E_{ki}	Equal to 1, if i is the exit switch for flow k
D_{ij}	Delay of the link between switches i and j
T_K	Maximum delay tolerated by flow k
P_{ij}	Maximum number of supported flows, if switch i runs service j
B_K	Bandwidth usage of flow k
H_{ij}	Capacity of the link between switch i and j
O_k	A range from 1 to length of service chain for flow k
O'_k	A range from 1 to length of service chain for flow k plus one
$V_{k_{l_{im}}}$	Is one, if the link between switches i and m is used, to reach to the l^{th} service in service chain of flow k
$A_{k_{ji}}$	Is one, if switch i processes service j for flow k

Table 2.1: Definition of variables of MILP formulation

in V. In this formulation we are minimizing the maximum utilization of the links and CPU cores of the network nodes (i.e., of the bottleneck). Core utilization is the number of flows using a CPU core over the maximum number of flows that can be simultaneously supported by a service on that CPU core. By minimization of the utilization, load is distributed more evenly in the network, avoiding hot spots and increasing residual system capacity. It also results in lower overall delay for flows.

Equation (2.1) is a constraint on the maximum number of services that can be supported on a switch. To avoid NUMA overhead, at most one service is assigned to a core. Consequently the number of services on a switch should be less than or equal to

the number of free cores on that switch. The next equation, Equation (2.2) reflects the process of selecting the necessary switches for a flow. M represents the available services on switches, and S stores the necessary services and their order of execution. Thus, X represents the possible switches for each order of execution of a particular flow (k). For each order, only one switch is needed for a service, but multiple choices may exist in X. Equation (2.3) selects one instance using the binary variable W. A limit on the number of selected switches is set in (2.4). Although for readability and clarity we show Equation (2.3) as a multiplication, it is not a non-linear constraint, because it can be re-written as follows:

$$N_{k_{il}} \leq X_{K_{il}}$$

$$N_{k_{il}} \leq W_{k_{il}}$$

$$N_{k_{il}} \geq X_{K_{il}} + W_{k_{il}} - 1$$

The legitimacy of this conversion is because both sides of the multiplication are binary variables. For the flow k, the selected switches for each order are stored in N. Entry and exit switches are added to the selected switches N, resulting in F as shown in Equation (2.5). For each order in flow k, the right hand side of Equation (2.6) is equal to zero except for the source (+1) and destination (-1) switches. The left hand side of this equation shows the difference between out-degree and in-degree of each switch, so each zero or one for V shows the selection of a link between two switches for a particular order and flow. To make sure that selected routes in Equation (2.6) are not very long and they do not exceed maximum tolerable delay for a flow, Equation (2.7) is used.

The maximum number of flows using a service on a switch simultaneously depends on the nature of a service such as the computation needed for that service, and the hardware

capabilities of the switch running it. P reflects this for each combination of service and switch. This may be specified a priori or obtained experimentally. Equation (2.8) stores the mapping between switches and services for the flow k in variable A. A is limited to the maximum number of flows defined for a service in Equation (2.9). Equation (2.10) and Equation (2.11) set the variable U to the maximum value of link or core utilization and finally Link utilization is enforced by Equation (2.10).

2.3.1 Variations in Formulation

The formulation above is the foundation, and we consider a few alternatives below.

Consider only Link or Core utilization

Equation (2.11) or Equation (2.10) may be omitted so that we seek to just optimize either the core or link utilization. This variation is helpful if one of the objectives, link or core utilization, has a high value and cannot be decreased at all. Hence it is better to just minimize the other one and increase the available capacity in the network.

Combined formulation with a penalty function for delay

The combined formulation, which covers link and core utilizations at the same time does not seek to minimize the delays experienced by flows, but just ensures the delay is below the tolerable value. After minimizing the utilization of the links and cores, it may be desirable to minimize the delay as well. For this, we can change the objective function to the following:

$$\text{Minimize: } U + MaxDelay/LV$$

LV is a large enough integer to reduce the effect of MaxDelay to an amount lower than the minimum variance of U. For example if the finest granularity of variance in U is equal to 0.01, LV may be two orders larger than the possible MaxDelay. Therefore the effect of the delay will be limited to one percent. With an equal value of U, the solution with smaller MaxDelay is chosen. MaxDelay, the maximum delay observed by a flow, is:

$$\sum_{l'} \sum_{i,m} V_{k_{l'_{im}}} D_{im} \leq MaxDelay \tag{2.12}$$

This constraint reflects the fact that MaxDelay is larger or equal to the total delay of any flow.

2.4 Developing simple Heuristics

The solution time for the optimal placement with the MILP grows exponentially with the number of flows, thus limiting the scale of the problem that can be solved. Moreover, once the optimal placement is arrived at, any changes in the set of flows or the assigned services at the switches requires the problem to be solved all over again, which may not be practical . We seek heuristic approaches to solve these problems.

Heuristic-A

This heuristic is a multi-step greedy algorithm without using the MILP. At first we select flows one by one and try to place their required services on free cores along their shortest path. In the next step, we seek to share the already assigned cores on the shortest path. In step three, we then look further at the neighboring switches and use their free cores to accommodate necessary services. In the next step we try to share already assigned cores

in the neighboring switches with flows whose requirements are not yet satisfied. If after all these previous steps a flow still does not have all the necessary services, Heuristic-A adds a node from the neighboring switches to the shortest path and repeats all the aforementioned steps.

Heuristic B, B+, and B+COR

For these heuristics, instead of solving the problem for all the flows at the same time, we divide the flows into the groups. We start from the first group, and solve the optimization problem for it. Based on the solution, the problem is updated again and we solve the updated problem for the next group. We continue this process until all flows are supported. We call this heuristic B. To be able to use information from a previous step, we have defined new set of variables and have changed some of the constraints of the MILP formulation. We call these variables $preM$, $preUL$, and $preUC$. The first, $preM$, represents the union of assignments of services to the cores in the previous rounds. The M variable in the formulation is replaced with $M + preM$ throughout. $preUL$ and $preUC$ represent the link utilization and core utilization respectively. $preUL$ is added to the left hand side of (2.11) and $preUC$ is added to the left hand side of (2.9).

In the MILP formulation we are not minimizing the number of used cores, so as a result some cores may be assigned but not used. To make this heuristic more efficient, we add a pre-processing phase which eliminates unused cores from $preM$. In the rest of the paper, this enhanced version is called B+. It is efficient in curing the both drawbacks of original formulation: if we do the processing in small groups, the problem is solved very

fast; and this method can be used to avoid solving the problem all over again. Just the problem for the added flows can be solved.

After studying the results with B+, we found that other methods of partitioning flows may help get better results. The most effective was $B + COR$. In first step, the shortest path between entry and exit switches for different flows is chosen. Then, the number of flows who have a common switch in their shortest path is counted and assigned to that switch. Then switches are sorted in ascending order based on the number of flows passing through them. Less crowded switches are selected first. The reason for efficiency of this algorithm is that trying to minimize utilization of bottleneck switches overuse lots of resources of neighboring switches in the hope of lowering the overall utilization. But if we place flows at crowded switches last, the necessary resources at neighboring switches are allocated and are used to satisfy the needs of hotspot flows. In other words, starting from the least crowded switches, helps us to have a more even distribution of resources in the network.

Heuristic C

The default formulation doesn't support minimizing the number of used cores, because the original formulation seeks to minimize the utilization at the bottlenecks. But with Heuristic C, similar to the B, flows are processed in groups. Therefore having more unassigned cores provides a greater opportunity for placing subsequent flows. It provides the flexibility to define the type of a service that has to run on a core for incoming flows. So Heuristic C is similar to B, except that the objective function is changed to: *Minimize U+ (MaxDelay / LV) + (TotalCores / VLV)*

The value of $TotalCores$ is calculated based on the following expression:

$$\sum_{i,j} M_{ij} \leq TotalCores$$

VLV is defined similarly to LV. The LV should be large enough that the maximum variance of TotalCores over LV should be smaller than minimum variance of MaxDelay over LV.

2.5 Evaluation

We evaluate the effectiveness of our MILP formulation on an example network topology, and initially use a default where all the flows have the same service chain comprising 5 services. All services support up to 10 flows on a single core, with the exception of $Service_4$ which only supports up to 4 different flows on a switch core. We use an off-the-shelf solver to solve the MILP and the non-optimization components are developed in Java. All the switches have homogeneous processing capability with 2 available CPU cores. The default topology in experiments is the network topology of AS-16631 from Rocketfuel [164] with 22 nodes and 64 links. We run a total of 12 experiments, with varying number of flows: 5 to 60 in steps of 5.

First we investigate different objective functions, $\_C$, $\_L$ and $\_M$, each minimizing CPU load, traffic load on the links or both CPU and link load, respectively. They can also minimize the maximum flow delay, after minimizing its main objective and this is represented by a $\_D$ suffix. To show that the optimization approach can support additional flows as they arrive, we generated random flows in groups of five each. We then placed these flows using the MILP with the different objective function. If the placement was feasible,

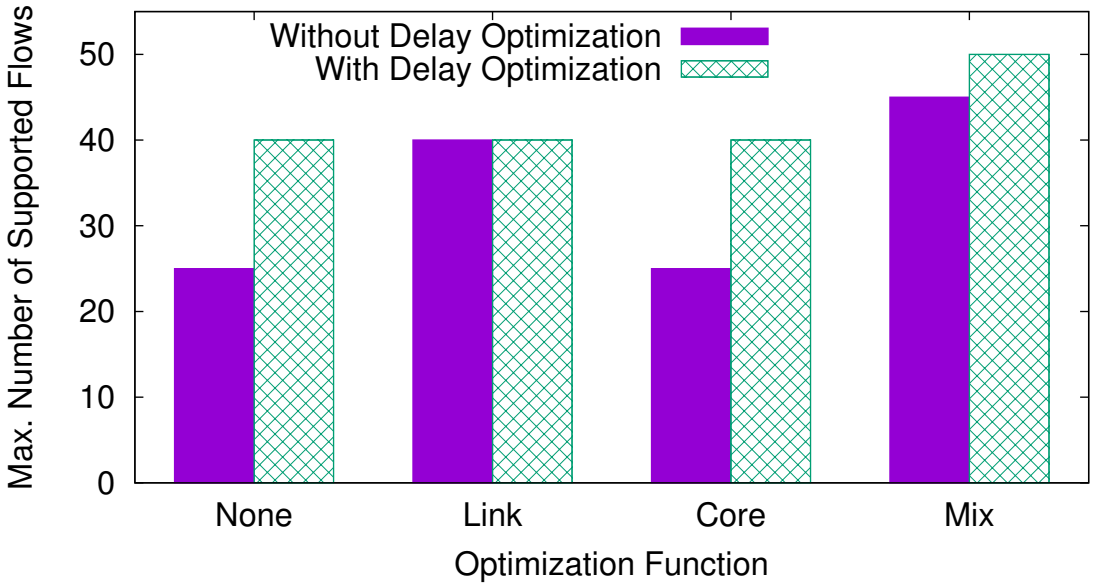


Figure 2.3: Maximum number of fitted flows for different objective functions

another group of five was added to the network. The results of this experiment is depicted in Figure 2.3. Each objective function with and without maximum delay minimization are shown as distinct bars. 'None' represents a placement without any optimization, and 'Mix' shows the results with the objective function $\_M$ combining link and core utilization. The significant improvement in capacity by minimizing delay in the 'None' and 'Core' cases shows that minimizing the maximum delay can be useful. However, minimizing the maximum link utilization is effective as seen with the $\_M$ (Mix) objective function, which has a higher capacity than both the 'Core' cases. Enhancements by adding delay minimization to the Mix case shows even better performance. Figures 2.4, 2.7 and Figure 2.8 show the maximum utilization of the most highly utilized core, link and the link or core for the solution with the 3 different objective functions $\_C$, $\_L$ and $\_M$. When only one metric (core or link

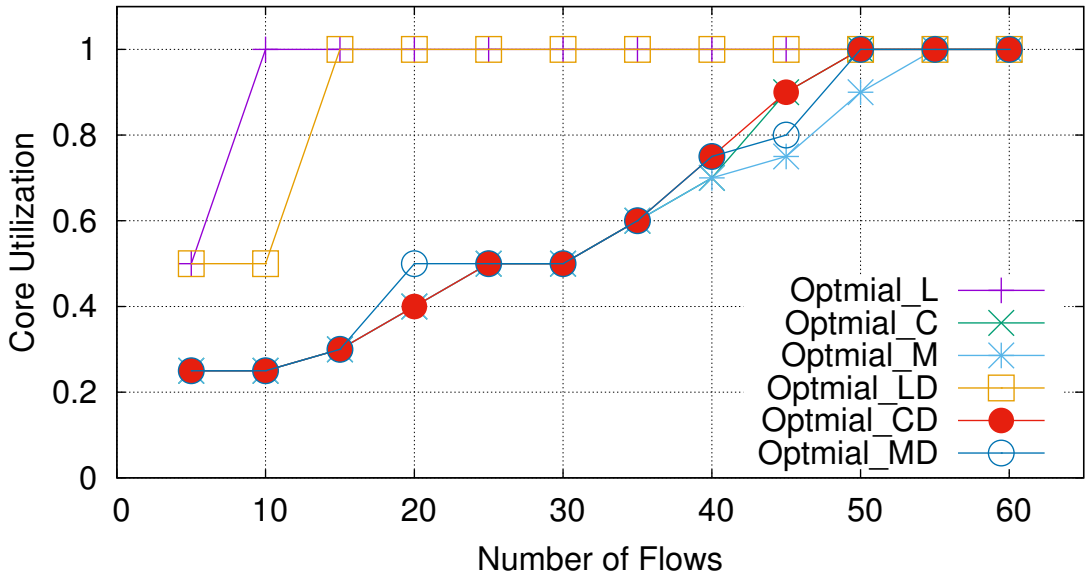


Figure 2.4: Max. core utilization for problems with different objective functions

utilization) is minimized, the utilizations of the other resource grows quickly, resulting in that resource also not being used by additional flows and reducing the flexibility for the placement algorithm. The combined objective function ($\_M$) is able to compensate for the shortage in one of the resources by increase in the usage of the other resource. The core and link utilization thus increase together.

As shown in Figure 2.8, when delay minimization is also used, we observe higher utilization compared to the case where there is no delay minimization (even though delay minimization is a second level optimization). This is because we only achieve a suboptimal MILP solution. In some cases, especially for large number of flows, the solver is not able to reach to the optimal solution even after running the solver for a long time. In these cases, we have reported the best achieved (suboptimal) solution for that problem. Figure 2.6 shows

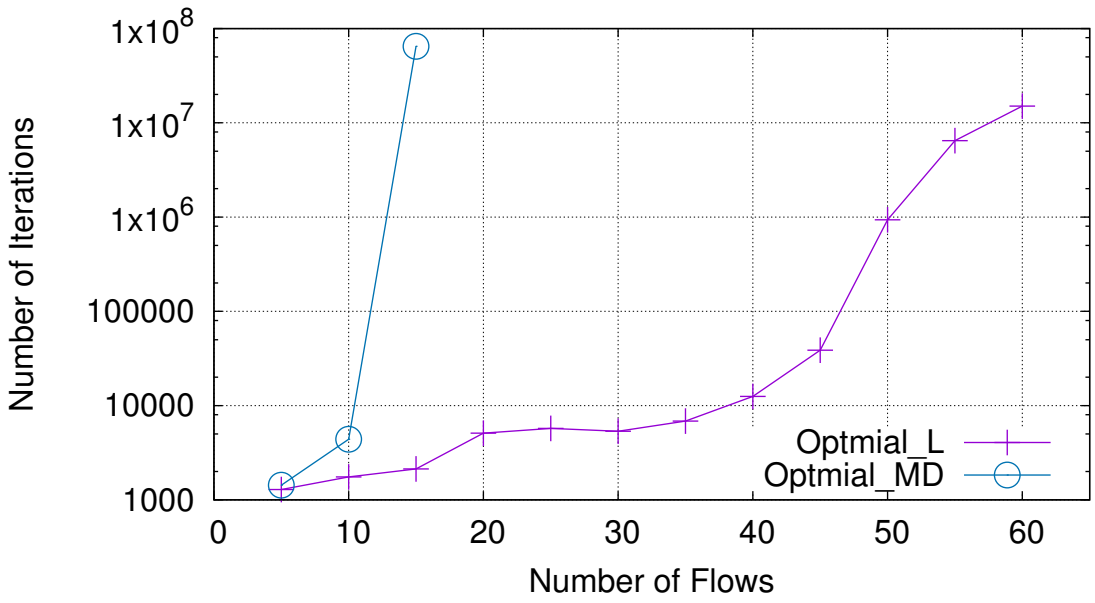


Figure 2.5: Number of solver iterations for different objective functions

which problems produced solutions within a "gap" percentage of the optimal solution. For example if we reached to a solution with an objective function with 17% higher value than the possible optimal answer, we mark it as 'solved' if the gap is 25%. It is marked as 'unsolved' for a gap of 0, 5, and 10 percent. The figure shows that the optimal solutions are reachable for all the cases up to 10 flows. However, we are not able to find the optimal solution even for 15 flows in some cases. This challenge is because of the exponential nature of this problem. The number of iterations needed for getting the optimal solution is shown in Figure 2.5. $Optimal\_MD$ needs substantial computation time even for 15 flows. As a result, we propose heuristics to overcome this scalability challenge. Figure 2.9 shows the

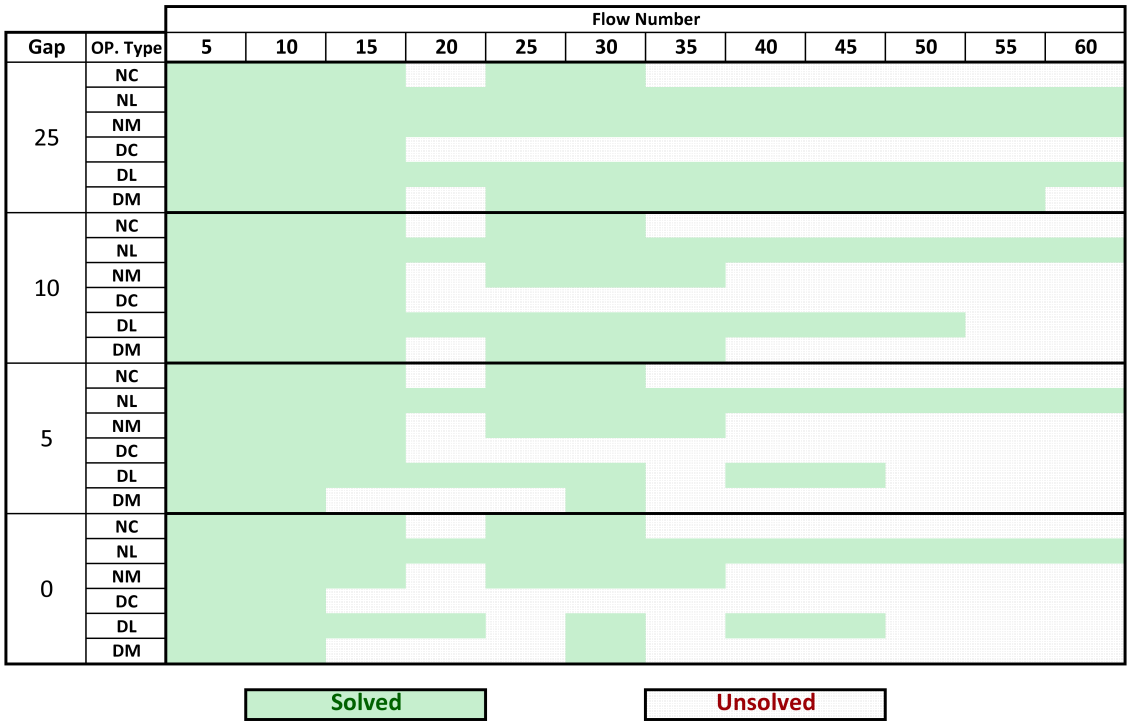


Figure 2.6: Status of found solution for each problem

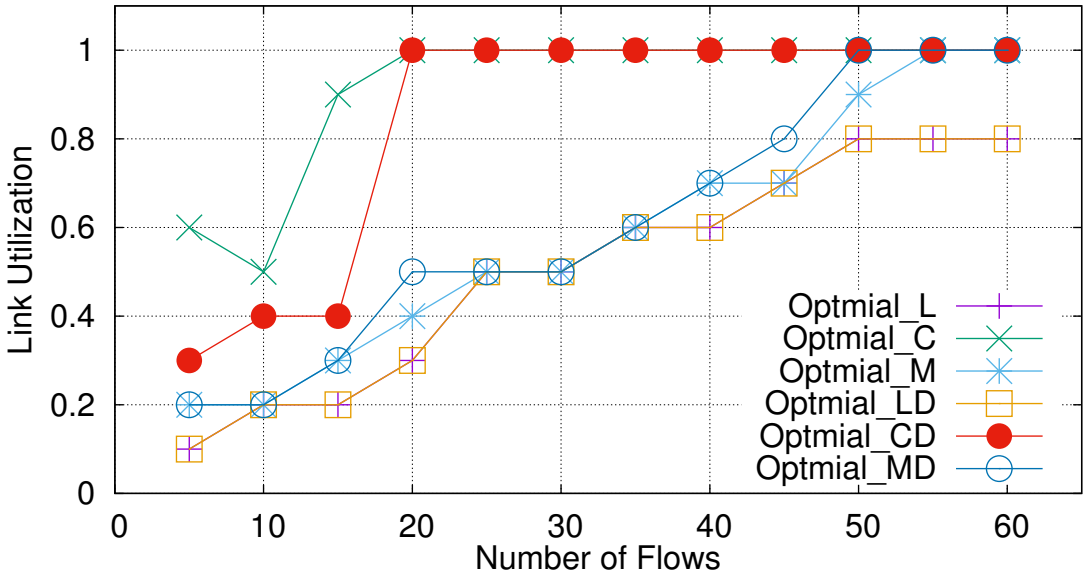


Figure 2.7: Max. link utilization for problems with different objective functions

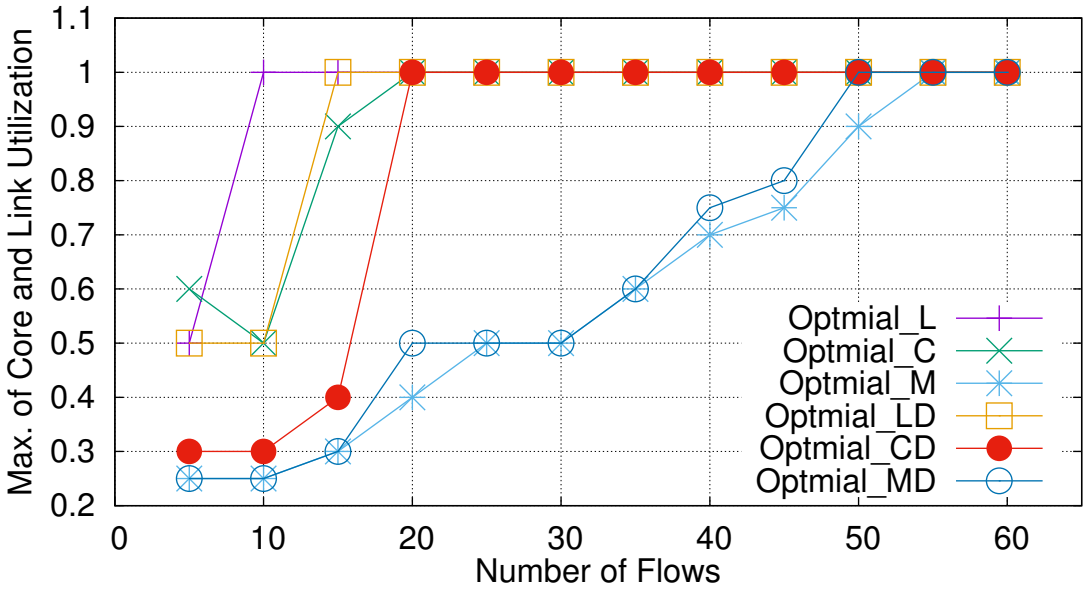


Figure 2.8: Max. core and link utilization for different objective functions

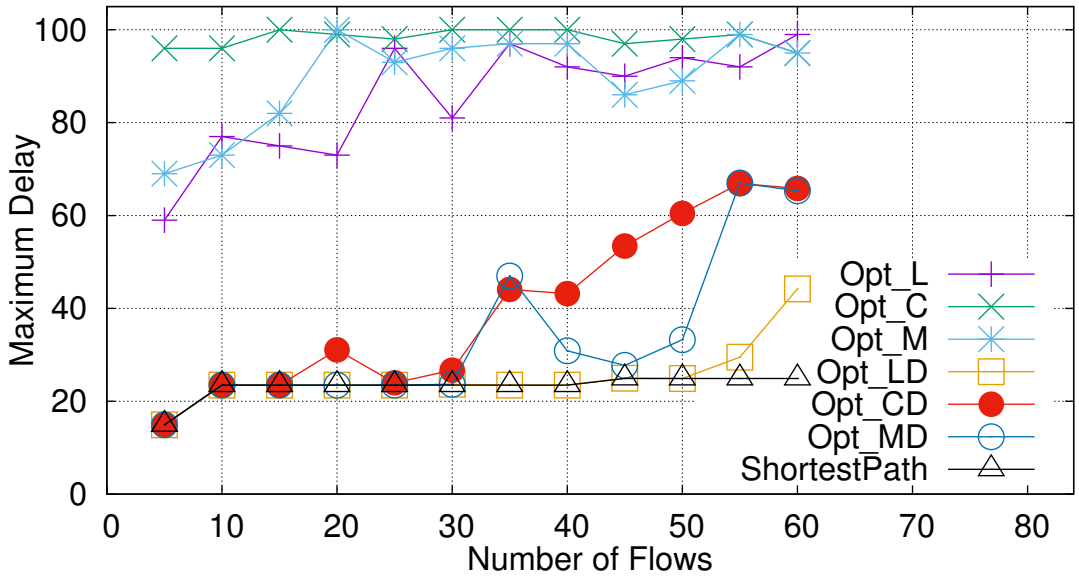


Figure 2.9: Maximum delay of flows for different objective functions

maximum delay of flows in the network, which is below 100 (the value set in the constraint for the maximum delay tolerated). Using the delay in the objective function does help in reducing the maximum delay of flows, and the maximum delay is less that three times the maximum delay of the shortest path even in the worst case.

2.5.1 Evaluation of Heuristics

Our main goal is to support as large a number of flows as possible in a network. The maximum number of supported flows by each method is illustrated in Figure 2.10. We are able to solve the original optimization problem, which returns the optimal placement, for up to 60 flows in this network. However, the time needed for computation of optimal solution in this network with 22 switches and 60 flows, is more than a day even on a server.

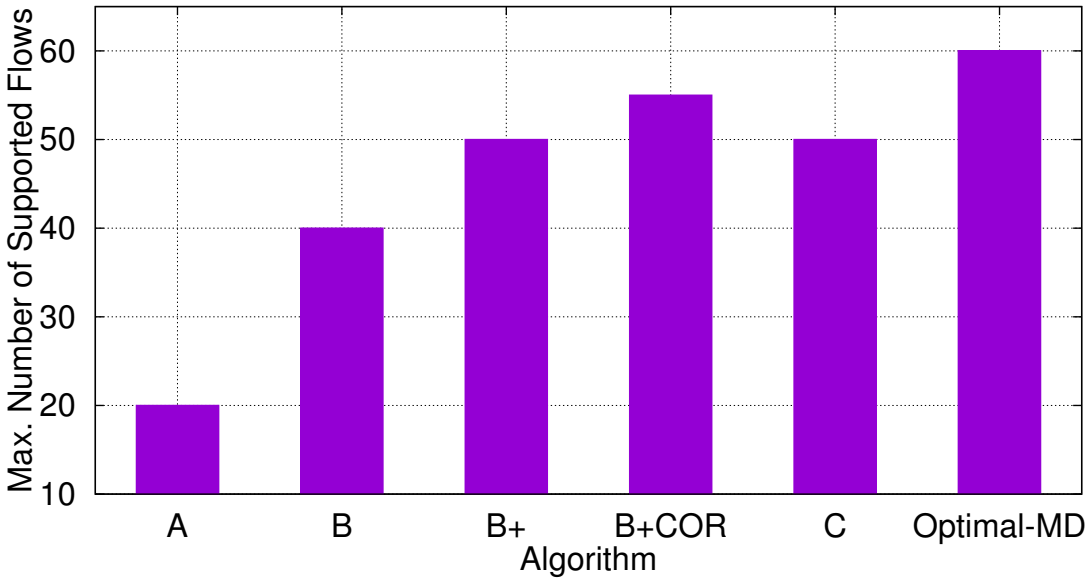


Figure 2.10: Maximum number of fitted flows for different algorithms

We therefore look at using the various heuristics described in the previous section. The most scalable method, $B + COR$, can fit 55 flows in the network, while solving the placement in a matter of seconds. Based on the bandwidth required for flow, the link capacities in the network and the maximum number of flows supported on each switch, the maximum number of flows that can be carried in this network is 60 flows, even with optimal placement and steering,. With our heuristics, we could fit up to 55 flows (92% of optimal solution). To show the scalability of our heuristic approach, we increased the capacity of network by a factor of 10 and then 100, by changing the available bandwidth and the flow capacity of each service. The resulting number of fitted flows in the network by heuristic $B+$ is shown in Figure 2.11.

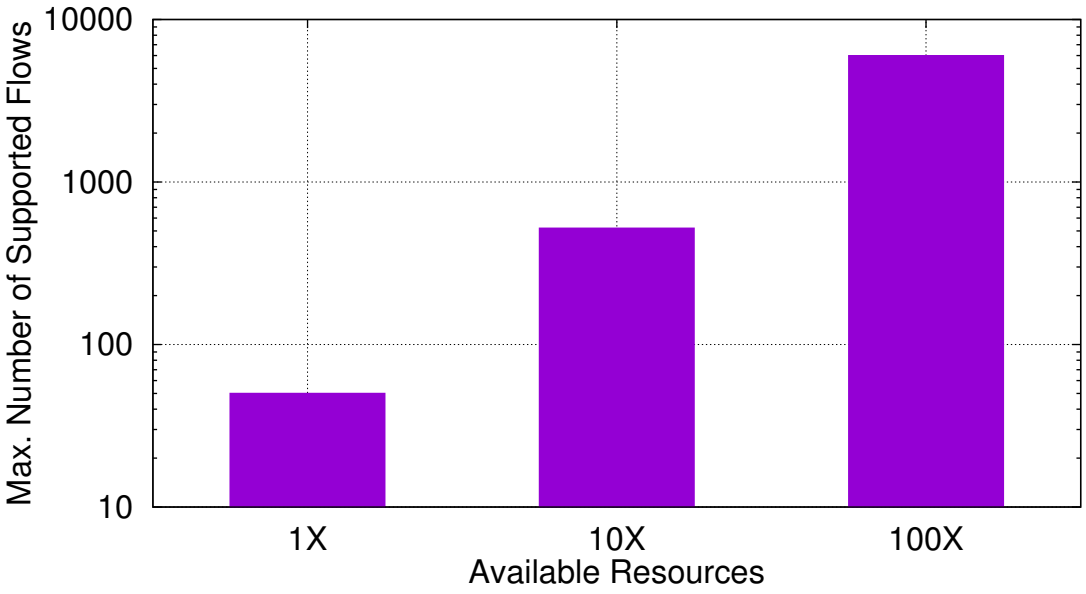


Figure 2.11: Maximum number of fitted flows for different network capacities

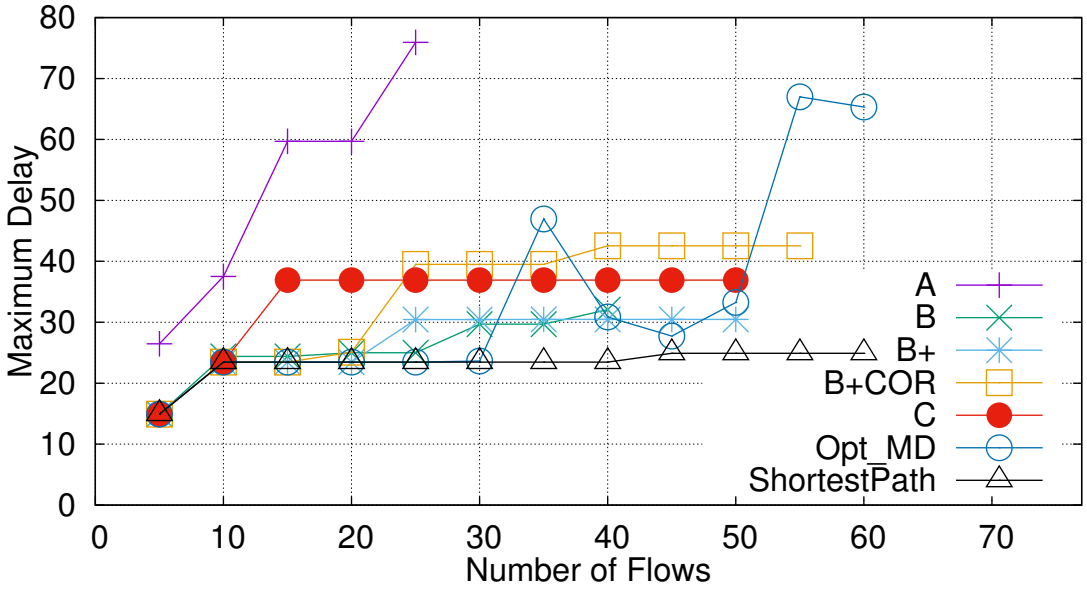


Figure 2.12: Maximum flows' delay for different algorithms

The comparison between maximum delay of algorithms and optimal solution is in Figure 2.12 shows the delay of the proposed heuristics is comparable to delay value of optimal solution (even lower in some cases). Since the optimizer first optimizes the core and link utilization and delay only as a second step.

2.6 Related Work

A large body of work exists for object placement and traffic steering. In our context, some recent approaches are:

CoMb[202]

CoMb is an architecture supporting consolidated middleboxes. Its optimization problem places services along the pre-defined paths of flows, leveraging the common parts of different services. One consequence is that it lacks the dynamic nature of placing NFVs in the network and the corresponding routing.

Stratos [81]

Stratos is an orchestration layer for virtual middleboxes in clouds. It uses an ILP formulation to decide how to steer flows through middleboxes, with a binary cost function between switches. Decisions about placement are made by an online rack-aware heuristic.

SIMPLE [185]

They use both an online and offline formulation. The main focus of the offline formulation is to keep limiting the size of forwarding rules due to the limits in the TCAM memory of SDN switches. The online formulation is for online load balancing on the available switches. However, it does not address the possibility of dynamic instantiation of services.

StEERING [243]

This work describes a system to dynamically instantiate a service at a desired network node and route traffic through such a service. The placement of the service is primarily through a heuristic.

T-Storm [232]

T-Storm is an online scheduler for Storm stream processing. There are some similarities between this scheduler and the placement and steering needed in an SDN-NFV network. For example, assigning executors to slots on workers resembles the service assignment to switches. However the proposed algorithm in T-Storm does not satisfy our needs as it primarily allocates executors based on the incoming traffic load and does not consider the network topology for decision making.

2.7 Conclusions

NFVs supported by an SDN protocol suite, provide a great opportunity to have higher level of flexibility and dynamicity in networks. However they introduce new challenges of joint service placement and traffic steering. In this paper, we provided a MILP formulation for this problem, which not only determines the placement of services and routing of the flows, but also seeks to minimize network link and network node core utilizations. We have devised heuristics to provide the opportunity to perform the placement incrementally without imposing a significant penalty. Our ongoing work is to enhance the scalability of our solution approach and to implement and demonstrate the use of the placement approach in practice.

Chapter 3

P4NFV: P4 Enabled NFV Systems with SmartNICs

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are transforming Data Center (DC), Telecom, and enterprise networking. The programmability offered by P4 enables SDN to be more protocol-independent and flexible. Data Centers are increasing adopting SmartNICs (sNICs) to accelerate packet processing that can be leveraged to support packet processing pipelines and custom Network Functions (NFs). However, there are several challenges in integrating and deploying P4 based SDN control as well as host and sNIC-based programmable NFs. These include configuration and management of the data plane components (Host and sNIC P4 switches) for the SDN control plane and effective utilization of data plane resources.

P4NFV addresses these concerns and provides a unified P4 switch abstraction framework to simplify the SDN control plane, reducing management complexities and lever-

aging a host-local SDN Agent to improve the overall resource utilization. The SDN agent considers the network-wide, host, and sNIC specific capabilities and constraints. Based on workload and traffic characteristics, P4NFV determines the partitioning of the P4 tables and optimal placement of NFs (P4 actions) to minimize the overall delay and maximize resource utilization. P4NFV uses Mixed Integer Linear Programming (MILP) based optimization formulation and achieves up to 2.5X increase in system capacity while minimizing the delay experienced by flows. P4NFV considers the number of packet exchanges, flow size and state dependency to minimize the delay imposed by data transmission over PCI Express interface.

3.1 Introduction

To overcome OpenFlow's limitations on being protocol specific and not being able to match on arbitrary packet fields [119], programmability using P4 [31] has been introduced. P4 enables simple networking devices to be programmed to support custom protocols and functionality [95]. This programmability provides flexibility to implement a custom networking stack and to quickly adapt (upgrade) to new protocols in the field. In addition, P4 being protocol-independent allows the implementation of custom switching pipelines and provides an API for the SDN controller to populate tables. Thus, P4 augments OpenFlow and extends SDN's capability to offer a new level of control and flexibility to dynamically configure and adapt the network forwarding plane to perform custom match-action processing.

To ease the burden on the computing resources (CPU processing, data movement) as network link bandwidths increase (going from 10 to 100 Gbps), smart Network Interface Cards (sNICs) are becoming increasingly common. Modern sNICs provide programmable multi-core processors that enable the host to offload custom-built packet processing functions dynamically. These include OpenFlow and P4 match-action processing capabilities. They also support custom NF processing that can be offloaded from the host to the sNIC [131]. However, effectively leveraging the sNIC capabilities and integrating the sNIC into the SDN control domain poses several challenges, namely i) Configurability: The hypervisor or the host software switch needs to be aware of the sNIC capabilities and provide the necessary flow routing configuration to ensure the traffic is routed appropriately between the host and sNIC through specific virtual (SR-IOV) ports. ii) Manageability: An SDN controller now has to control and program both the switch instances (in the host hypervisor) and the attached sNICs to configure the data plane operations. This increases the complexity of the SDN controllers. iii) Utilization: Since sNICs have limited computation and memory capabilities, not all NFs and switching can be realized on sNICs. Complex NFs, including stateful NFs, still need to be realized in the host. Further, chaining different NFs across the host and sNIC may result in a packet making multiple traversals across the host PCIe bus. This can result in high latency and excessive overhead [152]. To overcome these challenges, we have designed and implemented P4NFV, which offers a unified host and sNIC data plane environment to the SDN controller. Our framework takes into account both host and sNIC capabilities (viz., compute and memory resources), communication considerations (sNIC to host virtual port mappings, PCIe bandwidth, and delay)

and SDN policies (service chain configuration, service level objectives) to provide a unified view of a P4 capable NFV processing engine. The key contributions of P4NFV are:

- We create a DPDK [55] based P4 NFV framework to facilitate and configure SR-IOV based virtual ports for accessing the sNIC [129].
- We present a unified SDN-agent for the host and sNIC P4 switch. The SDN controller does not need to be aware of the differences between the host and the sNIC P4 capabilities.
- We provide an optimization framework for partitioning the P4 pipeline across the host and sNIC, accounting for the sNIC compute and memory constraints and the NF chain processing requirements.

3.2 Architecture

P4NFV is a framework that abstracts a server that benefits from a SmartNIC as a single entity to the SDN controller. In this framework, different Network Functions (NFs) that can be executed by the OpenNetVM NFV host [240] or the SmartNIC are sent to the SDN controller as possible external P4 actions which enables the central controller to shape the desired service chain within a P4 pipeline (hereon we use NFs and actions interchangeably) . When SDN Agent of this framework receives the pipelines configurations from the controller, it optimizes the placement of network functions and P4 tables between the host and sNIC. The decision is made locally, thus avoiding the overheads and complexity on the SDN controller. The design also minimizes communication between the host and

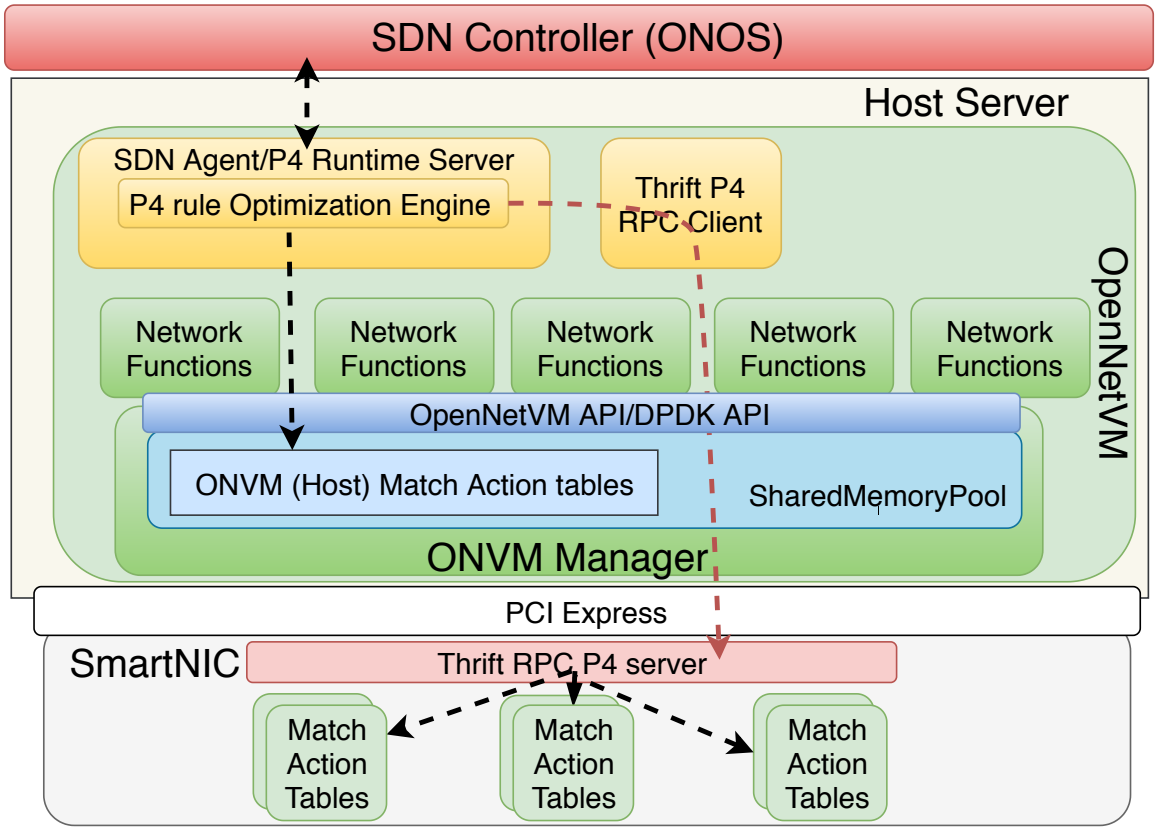


Figure 3.1: P4NFV Architecture

sNIC, thus judiciously managing PCIe bus overheads. The overall architecture is depicted in Fig. 3.1.

3.2.1 OpenNetVM

We build P4NFV on top of OpenNetVM [240], which is a scalable and efficient NFV framework that supports dynamic steering of packets through NF service chains. Routing packets to appropriate NFs is performed by an NF Manager in OpenNetVM, while packet data stays in shared memory. The NF Manager mediates packet access by NFs through packet descriptors. First, we extend OpenNetVM to support sNICs that

require the host device to interface with Single Root-I/O Virtualization (SR-IOV) based virtual ports (vport). OpenNetVM was initially designed with simpler DPDK-enabled NICs (that support multiple, 64-128 queues per port), where it maintains a fixed mapping between Ethernet port queues and the host Rx/Tx threads that poll one-or-more of the port queues. However, this design cannot support SR-IOV based vports, especially the Netronome sNIC [166] which limits to only queue per vport, entailing a strict restriction of 1 Tx and 1 Rx thread per vport. This greatly limits the throughput of the NF processing pipeline. To overcome this limitation, we design a scalable Tx Gateway component (thread with dedicated ring buffer) based on the Facade architectural pattern [78]. Tx Gateway thread can serve packets from different NFs and forward to dedicated vport queues.

Second, we extend OpenNetVM to provide the P4 switch capabilities conforming to the $P4_{16}$ switch specifications [33, 172], *i.e.,* include the P4 based tables (match-action processing pipeline) to process and route the packets. The p4 switch reads the input JSON file and generates the respective pipeline and tables. The progress through the tables is recorded by storing the pointer to the next table in the metadata of each packet. When a packet needs to be processed by an NF, the packet is sent to that NF's queue, receives the service, and then is processed in the switch by using the next table pointer. Hence, the p4 switch can keep processing other packets while a packet is processed in an NF. The p4 forwarding tables are stored in the shared memory and SDN agent can update the rules directly to lower update delay in the tables.

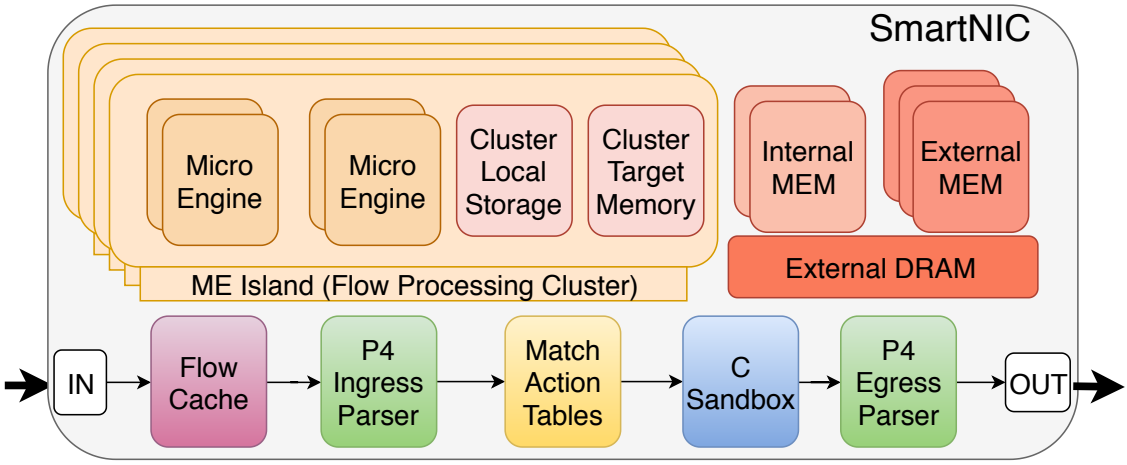


Figure 3.2: SmartNIC - Typical Blocks and Processing pipeline (based on Netronome NFP-Agilio4000) design [166]

3.2.2 Overview of SmartNIC architecture and capabilities

sNICs accelerate network processing by offloading some or all of the network protocol processing functions usually performed by the server CPU. Beyond the on-chip network acceleration capabilities of traditional NICs, for example, checksum, segmentation and reassembly, sNICS provide several programmable processing cores that can be used to implement complex network functions usually part of server processing.

We used the Netronome SmartNIC [166], as part of the P4NFV framework. The sNIC comprises of 60 flow processing cores also known as Microengines, running at 633MHz that is shared by eight threads running the same program. The sNIC also has a hierarchical memory subsystem, providing memory for a FlowCache, and custom user-defined data structures, and match action tables through memory transactions. Different memory subsystem components in the sNIC vary in capacity and also result in different access delays.

The sNIC Microengines can support MicroC programs that are written as actions in the packet processing pipeline as well as separate functions that can be executed asynchronously, invoked *e.g.,* by timer expiration. Typically stateful actions are carried out in MicroC. This allows the programmer to select the appropriate memory subsystem on the sNIC for custom data structures. To fully utilize the sNIC, the host-resident SDN agent must be aware of the capabilities of the attached sNIC, including partitioning tasks between P4 packet pipeline vs. MicroC programs, and then place data structures on on-chip (of the order of MBytes) memory vs. DRAM.

We focus on providing SDN support for sNICs, paying attention to the modification of rules in the P4 match action table, and handling the effects of caching those rules in a FlowCache. While the logical P4 pipeline is mapped to the sNIC and the table entries drive the selected action set, a FlowCache, similar to the Unified Flow Table fast path [76] and OVS megaflow [180], is employed to achieve high throughput. Since the sNIC supports P4, it allows for programmable parsing, including the definition of new headers. The packet processing cores conduct parsing (or header extraction) along with checksum verification. Subsequent packet processing is load balanced across available flow processing cores (Microengines) for match action processing. Based on the actions selected, packets may be DMA'd to the host using the PCIe DMA or sent to the egress packet processing cores that deliver the packets to the MAC buffer.

Typical flow processing pipelines involve multiple table lookups (hash key and index computation) for each packet at each stage of the table pipeline and executing the specified actions. These activities are generally repeated for subsequent packets. To acceler-

ate the data-path processing, sNICs can also include a dedicated cache (called a FlowCache) to store the flow lookup information. However, using such a FlowCache comes with its limitations. The caching of packet processing actions is not suitable for the cases where the action set for different packets of the same flow can differ (such as sampling, when a timer or a metric of interest reaches a certain threshold) because only the first packet's action-set is applied to all the subsequent packets.

3.2.3 SDN Agent on the host

We construct an SDN agent in the host to communicate with the central SDN controller, presenting a unified view of the host-based P4 switch and the P4 switch in the sNIC. As shown in Fig 3.1, SDN agent is a module in the OpenNetVM P4 switch and maintains the host P4 tables in shared memory which is accessible by P4NFV P4 switch.

Upon startup, the SDN Agent gathers information on the capabilities of the sNIC and the software-based OpenNetVM P4 switch and provides it to the SDN controller. The controller uses the P4 Runtime interface to sent the unified packet pipeline to the SDN agent. The SDN Agent also receives the set of tables and actions proactively for expected flows. When the SDN agent receives this information, it uses its optimization engine to decide the placement of the NFs (P4 actions) and division of tables between the host and sNIC. The optimization engine (see Section 3.3) seeks to minimize overall packet latency by accounting for the data dependencies, whether the functions best execute at the sNIC or the host and provides the SDN agent the information on the tables to populate on the host and the sNIC (some tables of a pipeline may be on both the host P4 switch and the sNIC). While the SDN controller views the host and sNIC as a unified packet processing

pipeline (and associated tables), the SDN agent will need to update the rules appropriately on the host and the sNIC based on its initial partitioning. This may require inserting or updating additional rules in the intermediate table entries, or tagging additional state information through the chaining of multiple actions for same rules across the sNIC and host P4 tables. *E.g.*, suppose we have the rule in Table 3 (on both host and sNIC) to send all the TCP packets with destination port 80 to a firewall (FW) NF (*i.e.,* FW is run on both sNIC and host). The optimization engine may have partitioned the FW functionality so that those packets that need to undergo deep packet inspection (rule in Table 4 of host) - a compute intensive function run only on the host, are sent to the host through an entry in Table 2 of sNIC, while rest of the packets are processed within the sNIC. Thus, in this case, the SDN agent will have to add new rules for sending the subset of the flows to the host by adding an entry into Table 2, with a corresponding action to DMA the packets to the host. However, these rules need some changes before being applied on the tables of the OpenNetVM P4 switch or the tables on the SmartNIC because the rules are sent based on the original pipeline and tables. The SDN Agent finds proper tables based on the assignment it has done before and update the rules accordingly. The act of sending packets to the other device can be done by using intermediate tables between original tables and add an action to send to the other device or by defining a specific action in p4 architecture.

Also, when the SDN controller seeks statistics from different tables, the SDN Agent does the aggregation of the results obtained from both devices (host and sNIC) before sending to the controller. Note: it is possible to infer such required state dependency based on the P4 code provided by the SDN controller. Overall, the SDN Agent ensures the

different switching components (host P4 switch and sNIC) are opaque to the SDN controller and optimizes resource usage to reduce the packet latency by intelligent assignment of NFs and tables to different P4 switches within the node.

3.2.4 Interfacing with SmartNIC

The SDN Agent administers rules for the sNIC's P4 Match action tables via a Thrift Remote Procedure Call (RPC) interface. As Apache Thrift [141] is a cross-language code generation engine, we generate a Thrift client in C and integrate it with the Host's SDN Agent to add, delete, edit, and retrieve table entries. To add a table entry to the sNIC, the Thrift client first creates an instance of a table entry object. The client then populates the instance's match attribute with a JSON string that indicates which packets should match this rule and the action attribute with a JSON string that indicates the action to undertake and the corresponding arguments to pass to the action subroutine. The client must also specify whether this is default rule and the rule name. Once the object is constructed, the runtime API is invoked with the table ID and entry instance as arguments. The API runs over Thrift's binary protocol with a blocking socket I/O protocol.

3.3 Optimization of task assignment between Host and SmartNIC

The optimization engine is a subcomponent of SDN Agent, responsible to efficiently partition and assign packet processing tasks across the host and sNIC P4 switches. It accounts for the forwarding latency, packet processing cost, PCIe bus bandwidth, and packet

data dependency constraints. We formulate a Mixed Integer Linear Programming Problem (MILP) to guide the partitioning of the pipeline and functions between the host P4 switch and the sNIC. Overall the optimizer has the following features:

- Considers the delay caused by the number of PCIe traversal
- Considers the update rate of the rules governing different flows (sNIC update rate is limited)
- Considers the data dependency and delay caused by the amount of data exchanged over PCIe.
- Satisfying the maximum tolerable delay of each flow and minimizes the overall delay of the flows.
- Decides the place of different actions and path of the flows.
- If possible, runs NF clones on both (Host and sNIC) devices.

We assume N flows are being served. Our objective is to minimize the total delay observed by all the flows:

$$Obj : \sum_{f=1}^{N} TotalDelay_f = \sum_{f=1}^{N} (ProcessingDelay_f + PCIFixedDelay_f + PCITransDelay_f)$$

Each packet is being processed through a number of tables (t) of the pipeline, where L denotes the length of the pipeline. C_a^H & C_a^S denote the time needed to perform action (a) on the packet by the host and sNIC respectively. Note that the time for performing

different actions vary and the time for the same action when performed on host differs from the time it takes to perform the same action on sNIC. *E.g.*, large state manipulation is faster on the host, while simpler actions like forward to next table are faster on sNIC. If only primitive actions are being performed on the packet, then it is one of the special cases of the actions which the delay is dominant by the forwarding delay on the tables for that device. The delay for forwarding on each table is represented as T^H and T^S for the host and sNIC respectively. If the device does not support a specific type of action, then the delay for this action can be defined as infinity. Finally, Si, j, k shows the action k is applied for flow i in table j and A shows the total number of actions. We define the Processing delay as:

$$ProcessingDelay_f = \sum_{t=1}^{L} \sum_{a=1}^{A} D_{f,t} * S_{f,t,a} * (C_a^H + T^H) + (1 - D_{f,t}) * S_{f,t,a} * (C_a^S + T^S)$$

$D_{f,t}$ is a decision variable that determines that what device serves table t for flow f. If $D_{f,t}$ is one, it is served by host, otherwise it is served by sNIC $((1 - Df, t) = 1)$.

$$D_{f,1} = D_{f,L} = 0$$

Each traversal over the PCIe imposes additional delay on the packets. This delay is reflected in $PCIFixedDelay_f$.

$$\forall f \in 1..N, \forall t \in 1..(L-1):$$

$$D'_{f,t} = D_{f,t+1} - D_{f,t}$$

$$D''_{f,t} \geq D'_{f,t}\ ,\ D''_{f,t} \geq -1 * D'_{f,t}$$

$$U_f = \sum_{t=1}^{L-1} D''_{f,t}$$

$$PCIFixedDelay_f = U_f * E$$

To achieve the delay caused by $PCIFixedDelay_f$, the number of PCIe traversal is needed. As a first step, we define D'. This helping variable shows the changes in the serving place of the flow, from sNIC to host (it will be equal to 1) and from host to sNIC (with value -1) or showing no change with value 0. To count the number of traversals, we need to the absolute value of this variable and a summation over the number of ones in this variable. However, using absolute value function makes the problem nonlinear, which is not desired. To avoid using absolute value function, we used D'' binary variable. By having the introduced constraints, whenever D''s index is one or minus one, D'' will be equal to one for those elements. The only other problem is that other elements can be either zero or one, so an over reporting on the number of PCIe traversal can be expected. However, this miscalculation will not happen, as we are minimizing the overall delay and optimizer will choose zero instead of one for non-constrained elements to keeps the number of PCIe traversal minimum.

In addition, to the fixed delay for crossing the PCIe interface, we also add a term

for the delay caused by the transmission over this interface, which is dependant on the data size.

$$PCITransDelay_f = \sum_{t=1}^{L-1} (P_f + M_{f,t}) * D''_{f,t} * E^D$$

$M_{f,t}$ shows the amount of state needed in table $t+1$ for the flow f from previous steps. For example, forwarding data in this step may be depending on the metadata stored in the previous table. P_f is the average packet size for this flow and E^D is the delay in microsecond per byte for the data transfer over PCIe.

So far, we introduced constraints which are related to the objective directly. Hereon, we cover the constraints used to provide the correctness of the solution. The first constraint in this group is to assure the PCIe bandwidth is not violated.

$$B^E \geq \sum_{f=1}^{N} \sum_{t=1}^{L-1} ((P_f + M_{f,t}) * D''_{f,t}) * B_f / P_f$$

In this constraint, we adjust the bit rate of the flow (B_i), base on the overhead of necessary state (M) and calculate the overall bandwidth usage of the traversal between sNIC and the host. The summation over all the bandwidths of the flows, should be less than the bandwidth of the PCIe (B^E).

In addition to the bandwidth limit of PCIe, each flow is required to meet its maximum delay requirement, which is received as input:

$$TotalDelay_f \leq DelayRequirement_f$$

Each action instance can serve a limited number of flows:

$$ActHCapacity_a \geq NumActH_a = \sum_{f=1}^{N} \sum_{t=1}^{L} D_{f,t} * S_{f,t,a}$$

$$ActSCapacity_a \geq NumActS_a = \sum_{f=1}^{N} \sum_{t=1}^{L} (1 - D_{f,t}) * S_{f,t,a}$$

Each device can accommodate several actions at the same time. For example, in OpenNetVM, it is suggested to dedicate a CPU core to each NF. Hence, the number of actions can be limited to the number of available cores. To count the number of actions running on each device, we need a binary representation of the above variable to be able to run summation over them. To convert these variables ($NumActH\&S$) to binary variables ($IsActH\&S$), we define the first four of the following constraints. The next two constraints are for applying the maximum number of actions that can be running on the host (R_H) and the SmartNIC (R_S). Finally, the last one does not allow to have multiple instances of the action on two devices if that action is not cloneable. The values for $NotCloneable_a$, a binary vector, is provided by the system admin. If shared state between different flows is used in an action, we cannot have multiple instances of that action. If an action is stateless

or its state is only related to the flows currently using the action, it is cloneable.

$$NumActH_a \geq IsActH_a$$

$$NumActS_a \geq IsActS_a$$

$$NumActH_a \leq IsActH_a * N$$

$$NumActS_a \leq IsActS_a * N$$

$$R_H \geq \sum_{a=1}^{A} IsActH_a$$

$$R_S \geq \sum_{a=1}^{A} IsActS_a$$

$$1 \leq (IsActH_a + IsActS_a) NotClonable_a$$

Another consideration for deployment of p4 tables on the host and sNIC is maximum rate of the updates that can be handled by each device. Especially in our experiment the maximum rate for the rule update on sNIC was limited. To ensure that device will not be overwhelmed by the amount of rule updates, following constraints need to be considered:

$$MaxHostUpdates \leq \sum_{f=1}^{N} \sum_{t=1}^{L} ExpUpdates_{f,t} * D_{f,t}$$

$$MaxSnicUpdates \leq \sum_{f=1}^{N} \sum_{t=1}^{L} ExpUpdates_{f,t} * (1 - D_{f,t})$$

$MaxHostUpdates$ and $MaxSnicUpdates$ show the capacity of the devices in rules updates. $ExpUpdates_{f,t}$ shows how many updates we expect for flow f on table t, and as we explained earlier, decision variable D shows if host or sNIC is selected for serving flow f on table t.

3.4 Evaluation

We evaluate the P4NFV framework on a server with 40 Intel Xeon 2.20GHz CPU cores and 256GB memory and Netronome Agilio4000 CX Dual-Port 10 Gigabit sNIC.

3.4.1 sNIC and Software P4 Switch Performance

In the first experiment, we evaluate the elapsed time between the table entry modification request and when it took effect. The delay incurred before observing the impact of the new rule on the traffic can be found in Table 3.1. The updates are drastically slower on the sNIC, which limits the sNIC's ability to host flows with a higher rate of rule updates. In the next set of experiments, we demonstrate the processing delay for various NFs resident on the sNIC and host.

In Figure 3.4, we show that the processing delay experienced by packets in the host NFs are lower compared to that of the sNIC NFs. This can be attributed to the higher frequency of the CPU core that is 2.2 GHz vs. 633 MHz attained from sNIC flow processing cores. However, we observe the latency overhead of one round-trip over PCIe incurs approx. 15μs. Furthermore, the sNIC exploits parallelism by load balancing packets, for processing, on all of its 60 Microengines.

In Figure 3.3 we compare the total round trip time over the PCIe, observed from the sNIC, for varying burst sizes of 1500-byte packets. sNIC sends these batches of the packets and host returns the packets to the sNIC. We observe a linear relationship between the amount of sent data and the total round trip time. By assuming a linear trend equation

Operation	sNIC (μs)	Host (μs)
Insert	221 ± 11	0.42 ± 0.69
Edit	220 ± 18	0.31 ± 0.61
Delete	169 ± 20	0.31 ± 0.61

Table 3.1: Table entry modification response time on sNIC and P4 OpenNetVM switch

and using the least square method, the slope of 1.66 and intercept of -4.27 is obtained. These values are used to minimize the overhead of PCIe traversal in the optimization engine.

3.4.2 Optimization Engine

We used the Gurobi MILP solver to implement and solve the optimization formulation. In the first experiment, two of the actions have a limit on the maximum number of flows that they can support on the sNIC. The limit for the first action is 100 flows (stateless) and 300 flows for the second (stateful). The first, stateless, action can be scaled out as needed by having instances on the host as well as the sNIC. Increasing the number of flows from 50 to 400, we see that the delay for all the flows goes up, as seen in Fig. 3.5. With 50 to 100 flows, all flows are served by the sNIC. Above 100 flows, the instance for the first action reaches the sNIC limit. The overflow of flows is sent to the host. Thus, the delay increases more rapidly. Once we reach 300 flows, the limit for the second action on the sNIC is reached. Since this action cannot be split across the host and sNIC, we observe a sharp increase in the total delay. Beyond this, the delay increases depending on the service time for both actions to be executed on the host. Thus, the optimizer decides to forward packets between the sNIC and host intelligently as needed, leveraging on the ability to instantiate

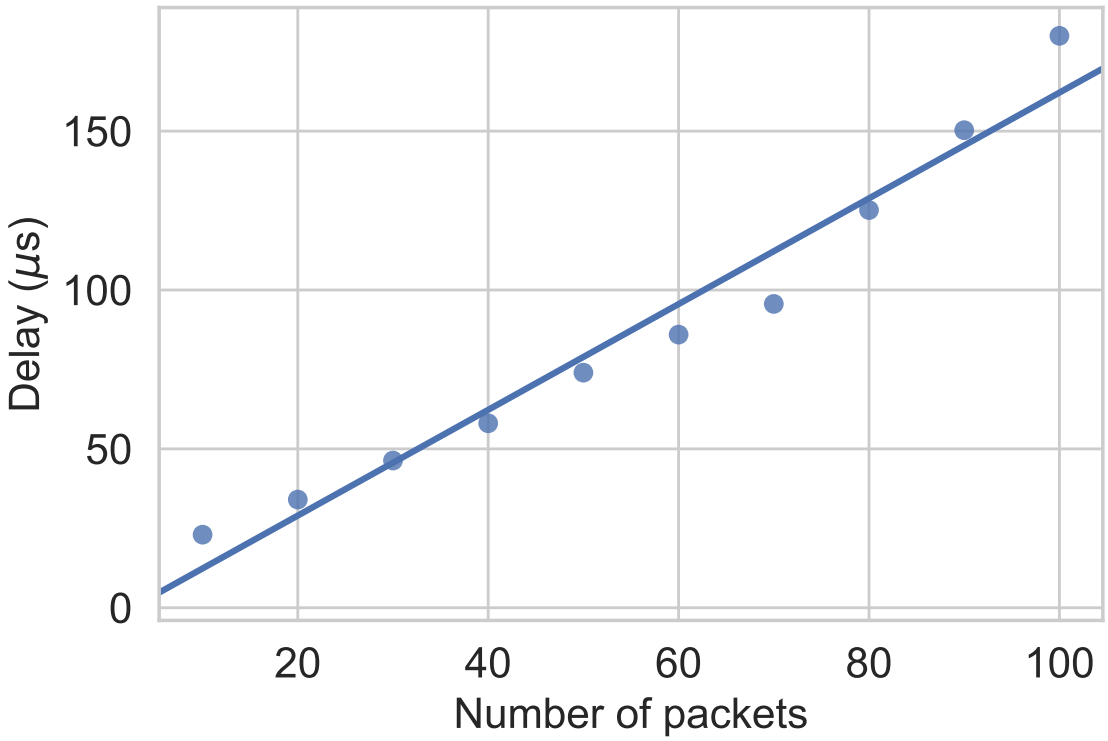


Figure 3.3: PCIe RTT measured from sNIC (varying burst size)

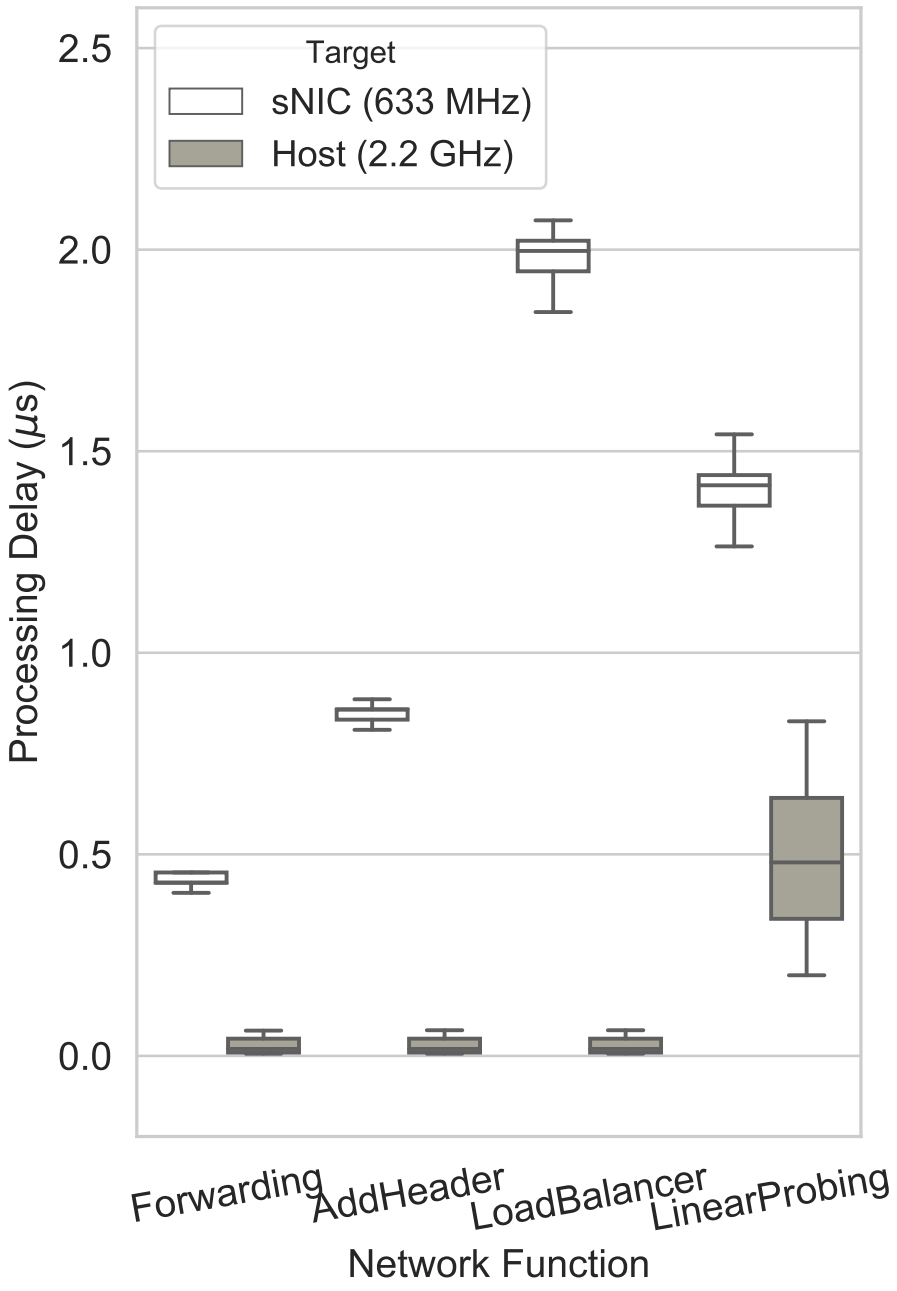


Figure 3.4: NF Processing delay

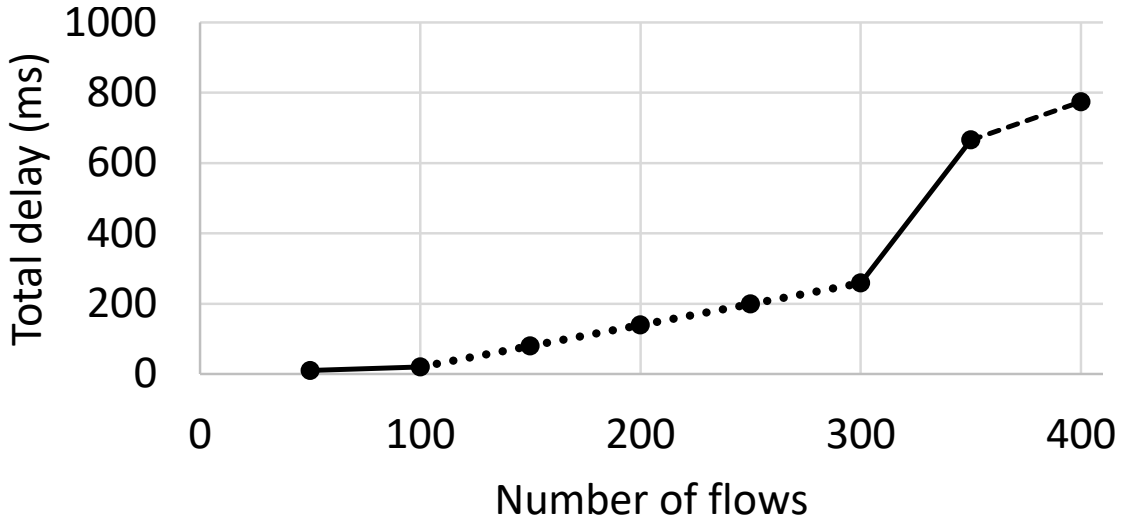


Figure 3.5: Total delay based on the number of flows

multiple instances of stateless NFs. It initially chooses to instantiate an NF on faster sNIC, but will switch to the host because of its higher capacity, while sharing the execution of actions across both when possible.

Since the rule update rate is lower on the sNIC compared to the host, we then measured the maximum number of flows that can be served in the network. For the update workload, we vary the number of updates for each flow using a uniform distribution varying from 1 to 10 updates per second for each table. Based on our measurement, the maximum update rate of the sNIC is set to 10000 updates per second based on the order of 100 micro seconds to update the rules in tables of sNIC (Table 3.1). The result for this experiment is illustrated in Fig. 3.6. It is possible to serve up to 2.5 times more flows if update rate is one of the considerations for the placement of the NFs and routing of the flows.

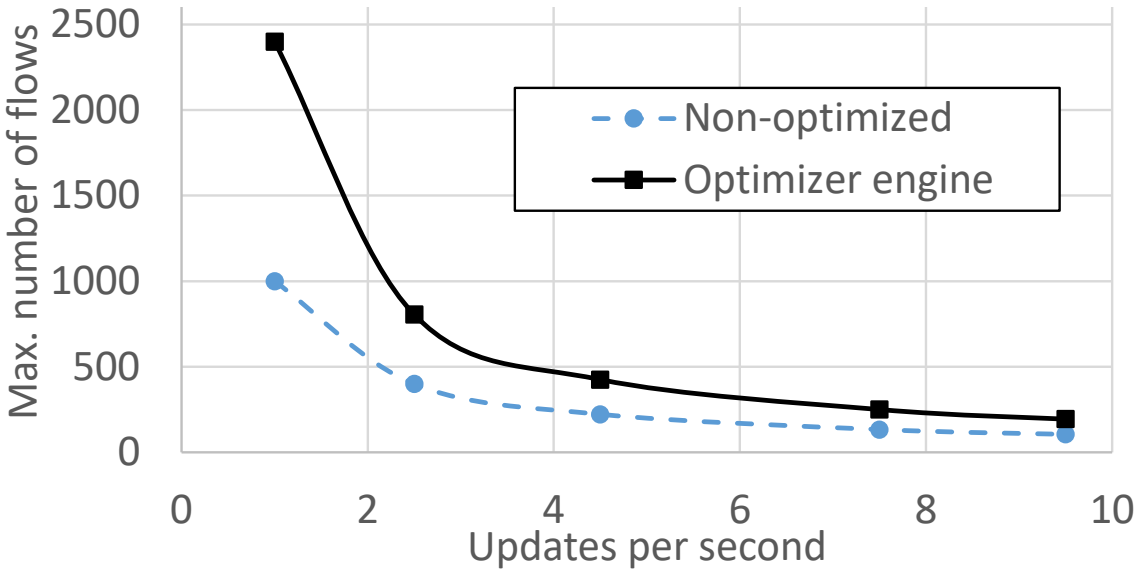


Figure 3.6: Maximum supported flows based on the update rate

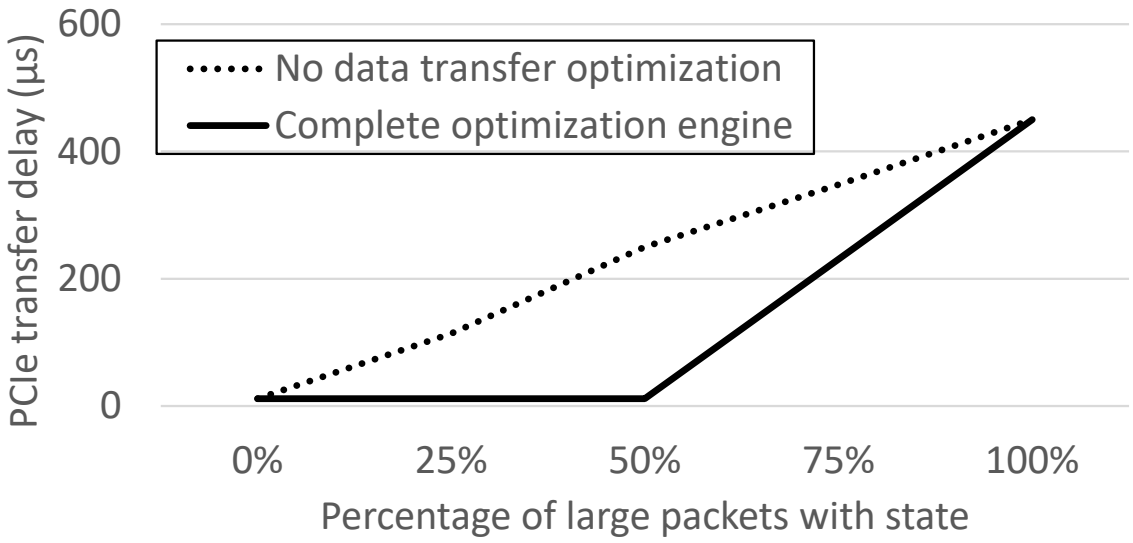


Figure 3.7: PCI Express transfer delay based on packet composition

As we observed in Fig. 3.1, the delay over PCIe is depending on the amount of data transmission. Fig. 3.7 shows if a number of the flows with different average packet sizes and data dependency need to be exchanged between the host and sNIC, the optimizer will select flows with smaller packet size and state dependency to traverse the PCIe. In this experiment, some of the NFs in the sNIC can handle half of the flows and packets of the rest of flows are sent to the host. In addition, packets are separated into two groups. In one group, packets are larger (1500 bytes) and they need 1kB of state from the previous tables. In the other group, packets are smaller (64 bytes) and they do not need any state from previous tables. By selecting the smaller packets with smaller data dependency, the optimizer reduces the delay imposed by PCIe transmission.

Finally, we compared the performance of the proposed optimization engine with one of the related work, UNO [131]. We implemented UNO's optimization formulation for

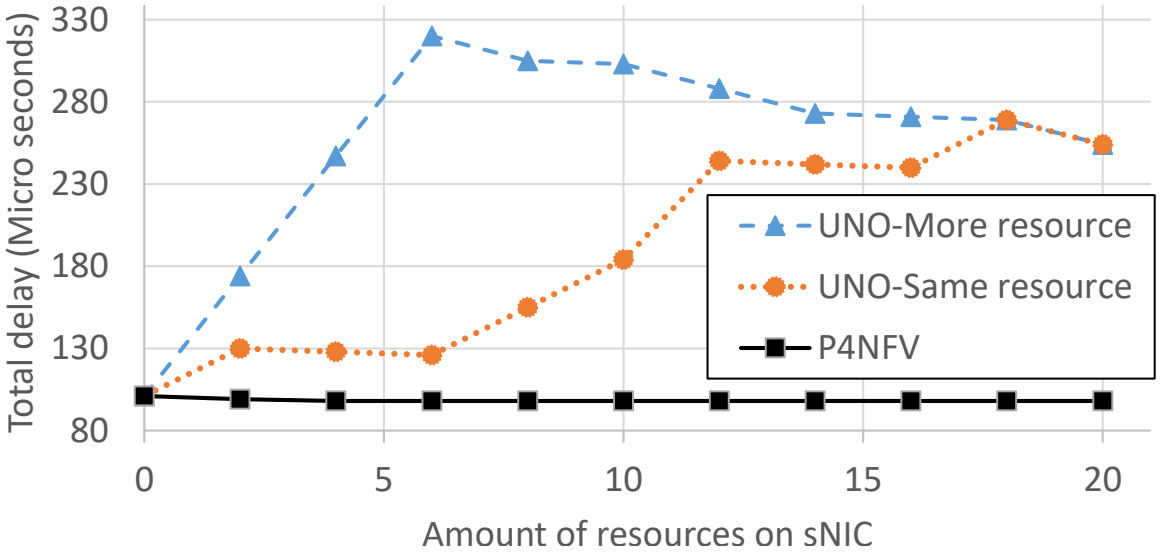


Figure 3.8: Comparison of observed delay of P4NFV Vs. UNO [131]

the same solver that we used to solve our problem. We considered a scenario that 20 NFs serve a single flow (UNO does not support multiple flows in its formulation). Six of these NFs are complex NFs which are faster on the host (10μs processing delay on the host and 40 μs on the sNIC, 4x difference is based on our observation in Fig. 3.4). Other actions are faster on sNIC (1μs on the sNIC and 2μs on the host). The result of this comparison is depicted in Fig. 3.8. We have considered two different scenarios for UNO. In *UNO-More resources*, complex actions need more resources in comparison to other actions and in *UNO-Same resources*, they use a similar amount of resources. UNO focuses on minimizing resource usage on the host, while our engine considers all the available resources on both host and sNIC, and uses resources on the host if they are available for different NFs. This difference allows P4NFV engine to minimize the delay caused by the processing of the packets and PCIe traversal.

3.5 Related work

sNIC as VNF accelerators and Optimization frameworks for offloading: Several works [131, 152, 162, 62] address NF acceleration by offloading NFs from the CPU to sNIC. Uno [131] presents an ILP based NF placement algorithm. They proposed an intuitive NF migration solution to alleviate overloads by identifying the bottleneck NF with maximum processing capacity and migrating it to CPU. However, this approach may adversely impact the latency of the service chain due to increased cross PCIe bus data transfers and not considering the differences in processing delays. In [152], authors propose the NF selection scheme, PAM, that tries to reduce the service chain latency by alleviating the hot spots on the SmartNIC by pushing only the border NFs (*i.e.,* either the start or end of chain NFs on sNIC) or their immediate neighbor NFs. They again seek to minimize the cross-PCIe transfers, but do not account for the difference in the processing delay of NFs, data dependency, or delay constraints of individual flows that we account for in our work. Also, the details on supporting the SDN controller to set up rules on sNIC and host are not considered. Unlike offloading an entire NF, authors of [162] identify a set of candidate operations *e.g.,* TCP connection setup and teardown, connection splicing and packet filtering and coalescing operations for stateful middleboxes that can be dynamically offloaded to programmable NICs, and gauge the expected benefits in terms of CPU load reduction, throughput, and latency improvements.

Virtualization on P4 platforms: The focus of another body of the work, such as [96], [95], [236] , and [237] is on providing the virtualization on P4 platforms. They facilitate serving multiple pipelines on a single P4 switch. While because of the virtualization, they

are close to the area of this paper and our framework can benefit from these approaches to support multiple pipelines in parallel, the focus of this paper is on designing an efficient sNIC enabled P4 switch which differentiates the focus point of this paper. Another contribution of the studies, such as [236] and [237], is providing uninterrupted reconfigurability of data plane in P4 switches. Similar techniques are applicable on P4NFV as well.

3.6 Conclusion

We identified key challenges (configuration, management, and utilization), in incorporating P4 capability into NFV platforms, including both the host and sNICs. We propose P4NFV as a framework to provide a unified P4 data plane component, including both the host and sNIC P4 switches that can be managed by an SDN controller. P4NFV incorporates local intelligence to efficiently partition the packet processing tables and NF functionality on both the host and sNIC. P4NFV accounts for both networking *i.e.,* bandwidth, workload and packet forwarding latency of the unified host device (sNIC and the host P4 switches) specific constraints in a MILP formulation. We consider the packet processing cost, PCIe bus bandwidth, and latency for crossing the PCIe boundary as well as packet data dependencies. We demonstrate that P4NFV can achieve 2.5x improvement in system utilization and throughput, and minimize overall latency by up to 4x compared to the placement engine of [131].

Chapter 4

Protocols to support autonomy and control for NFV in software defined networks

The use of Network Function Virtualization to run network services in software enables Software Defined Networks to create a largely software-based network. We envision a dynamic and flexible network that can support a smarter data plane than just simple switches that forward packets. This network architecture needs to support complex stateful routing of flows where processing by network functions (NFs) can dynamically modify the path taken by flows, without unduly burdening or depending on the centralized SDN controller. To this end, we specify a protocol across the different components of an SDN-NFV environment to support the creation of NFs required by a service graph specification, using an orchestrator speaking to an NF Manager running on each host. We take advantage of,

and extend, the concept of the SDN controller-to-node protocol (OpenFlow being the most popular) and tagging flows to support complex stateful routing. Output generated by NFs processing packets may be returned to the NF Manager to influence dynamic route changes based on a priori rules defined through a service graph specification provided by network administrators. We envisage the SDN controller setting up these rules based on the output from NFs, the flow specification as well as global tags. By not treating tags as an independent component for routing, we show that we can dramatically reduce the number of tags required across the entire network. Further, by providing the right autonomy in decision making at the NF Manager and the individual NFs in our hierarchical control framework, we significantly reduce the load on the SDN controller.

4.1 Introduction

In our SDNFV architecture [71], we use the SDN controller and the NF orchestrator, coordinated by an SDNFV Application to create a more flexible and dynamic network: the SDN controller manages the network control plane [74] and the NF Orchestrator manages NFV instances and their flow state [82]. The SDNFV architecture is consistent with ETSI NFV architecture [108], with some differences, as we extend the ETSI model's adherence to the stricter control plane - data plane separation espoused by the SDN architecture. For example orchestration roles of NFV Management and Orchestration unit is assigned to the NF Orchestrator and management duties are assigned to SDN controller. A placement engine decides how functions are instantiated across the network, and the NF VMs on each host are controlled by an NF Manager. This paper focuses on the protocols that

split control across this hierarchy—exploiting the application-awareness of individual NFs, the host-level resource management capabilities of the NF Manager, and the global view provided by the SDN Controller and the SDNFV Application.

One of the approaches for managing complex middlebox deployments considered in the previous work [185, 72] is to introduce additional information in the form of *tags* to mark packets processed by a middlebox, and use the tag to indicate the result of the processing to the SDN controller. The SDN controller may then use this information to re-route the flow, potentially on a path that might appear to have loops when viewed at the network layer, but is required to route the flow through different middleboxes connected to the same switch [185]. In the FlowTags approach [72], an NF first receiving a tagged flow will 'consume' the tag after contacting the controller to elicit its meaning. However, existing approaches result in significant overhead due to the need for frequent communication with the SDN controller to generate and interpret tags. Further, the amount of packet header space dedicated to tags may become unreasonable since a large number of unique tags are needed to differentiate each hop in the path, for all the flows in the network.

To achieve our goal of having hierarchical control where both a smarter data plane and a logically centralized SDN controller have the capability to route flows appropriately as well as dynamically instantiate functions in the network, we also leverage the idea of using tags as in FlowTags [72], but suitably enhanced. We use the tags only when their existence is crucial and a forwarding decision would be ambiguous without using the tags. We have designed a set of protocols so that NFs can impact the route for a flow (or a class of flows), as well as communicate information such as the flow's state to the higher levels

in the control hierarchy (the NF Manager and the SDN Controller). Our work makes the following contributions:

- A hierarchical architecture to efficiently and flexibly combine SDN and NFV.
- An approach for supporting diverse NF service chains that support dynamic routing of flows based on the output of the NFs, even for NFs changing headers such as a NAT or for NFs affecting upstream NFs in the chain.
- Leveraging an SDN controller's knowledge about network topology and NF placement to limit the use of tags only to essential cases, where routing is ambiguous without their existence.
- Protocols across the SDN/NFV hierarchy that minimize the amount of communication with the SDN controller by exploiting the intelligence in the software-based data plane.

We evaluate the benefits provided by our architecture in terms of reducing SDN control message overhead and the number of distinct tags (and thus the additional header space) required network-wide.

4.2 System Components

Our SDNFV architecture provides a hierarchy of control with NFs at the bottom and the SDNFV Application at the top, as illustrated in Figure 4.1. In this section, we explain the role of the components in the architecture.

SDNFV Switches: The commodity servers running NFs in our platform are called SDNFV Switches, since we view them as "smart switches" that can be deployed in the net-

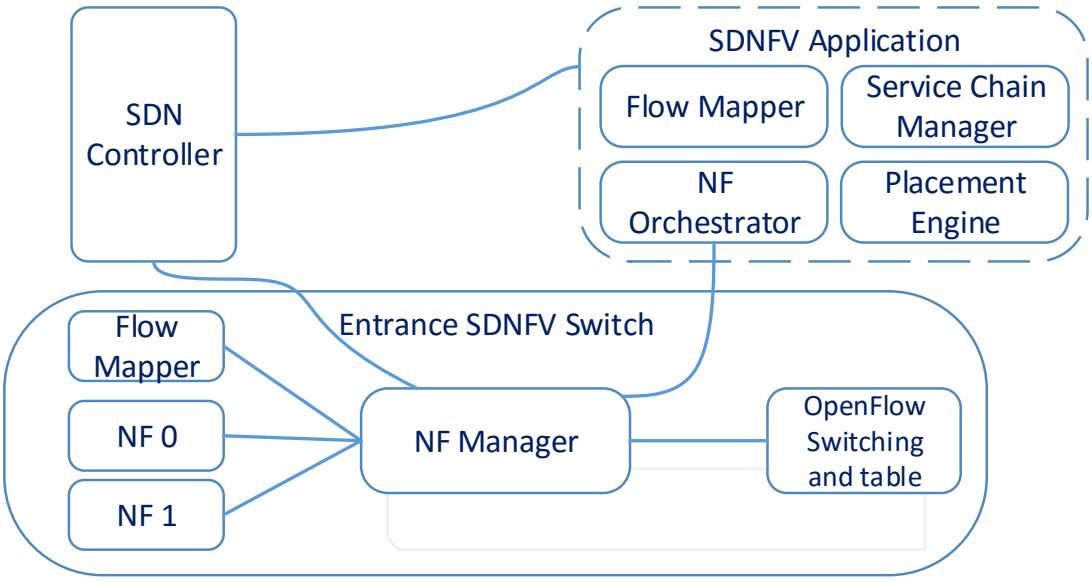


Figure 4.1: System components and their interactions (All the other switches are similar to the entrance switch but they do not have the Flow Mapper in them.)

work to not only forward packets, but also apply complex functionality to packet flows. As software based switching achieves near-wire-rate performance [193, 54]—even in virtual environments [98, 194]—the distinction of having NFs in COTS servers versus specialized switching hardware is likely to blur, and our SDNFV architecture recognizes this evolution. SDNFV switches interact with both the SDN controller and the NF orchestrator.

Network Functions (NFs)

Today's networks comprise many different functions beyond simple forwarding. These include middleboxes such as firewalls, proxies, network address translators (NAT), etc. In our architecture, NFs are able to generate an output when they process a packet. This output is send to NF Manager by using the shared memory between the NFs and the NF Manager.

NF Manager

Each SDNFV Switch may run multiple NFs coordinated by a single NF Manager on that host. The NF Manager is a software layer between the NFs and the operating system of the SDNFV switch. The NF Manager is responsible for managing NFs residing on the host (creating instances, forwarding packets to and between NFs), and controls the OpenFlow-like forwarding table. All the interaction to external network components (such as the SDN Controller, the NF Orchestrator) with NFs is mediated through the NF Manager.

SDNFV Application: At the top of the hierarchy is the SDNFV Application. This is where the network administrator specifies the overall application logic that will control

where NFs are instantiated and how packet flows will be routed through them. The SDNFV Application has four main components:

Service Chain Manager

This provides an interface to define sequences of NFs (e.g., middlebox functions) traversed in a specified order. These service chains can be dynamic, in that the output of an NF can result in forwarding the packet (or subsequent packets of the flow) to one of several different NFs. Thus, the chain is actually a graph with branches based on NFs outputs, rather than a linear sequence. Several example service chains are shown in Figure 4.2. Each service chain is given a *Chain ID* and a priority. When a packet flow matches the criteria for multiple service chains, the service chains are applied in priority order.

Flow Mapper

After defining a service chain, we need to assign it to the appropriate packet flows. In its simple form, the mapping may be predetermined by a network administrator based on the IP 5-tuple, including the wild-carding of some fields. For more complex situations, where the decision has to be based on the state of the flow (or related to multiple flows), the flow mapping may be done dynamically as packets arrive. This task is carried out by a flow-to-service chain mapping engine implemented as a part of the SDNFV Application or as one of the NFs in the entry switches. In the former case, new flows are sent to SDN controller and in the latter, the result of flow mapper NF is sent to SDN controller. Using

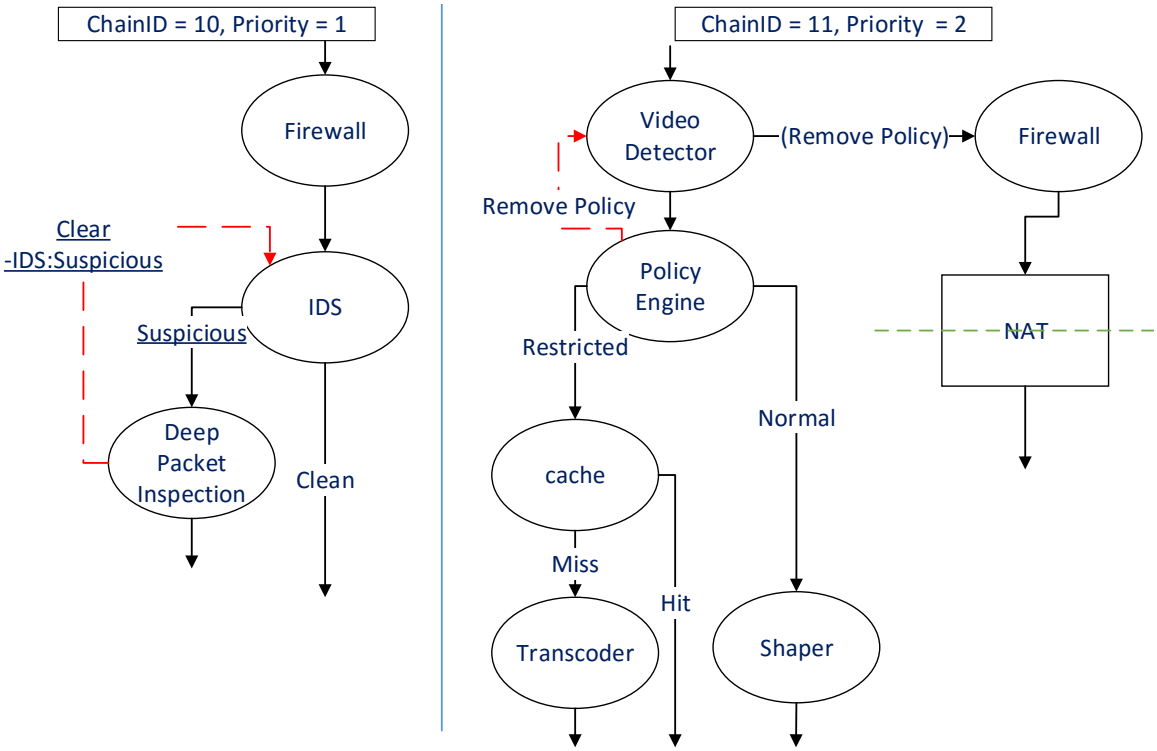


Figure 4.2: A sample cluster of service chains

NFs to perform flow classification, as shown in Figure 4.1, may be preferable if detecting the flow type is computationally intensive.

Placement and Routing Engine

The Placement Engine receives information about the flows, the service graph for those flows and the current network conditions (e.g., available capacity, topology) to decide where to instantiate NFs. We have proposed a mixed integer linear programming (MILP) solution for this complex problem, as well as heuristics to obtain results close to the optimal solution, in reasonable time [156]. The heuristics also enable us to solve the problem incrementally so that assignment of NFs to already established flows are not impacted by new incoming flows.

NF Orchestrator

The NF Orchestrator receives instantiation or migration requests from the Placement Engine (e.g., when a new service chain is deployed) and from NF Managers (e.g., when a server is becoming overloaded and a new replica NF is needed). It then deploys new VMs to run the appropriate functionality. It also handles the movement of state and syncing of the state between different replicas of an NF in conjunction with the respective NF Managers [82].

SDN Controller: The SDN Controller learns of the overall network goals from the SDNFV Application via its northbound interface, and communicates with NF Managers via its southbound interface. The protocol between the controller and the SDNFV switch needs to be enhanced, so that stateful control can be exercised by the NF Manager.

As is becoming the common practice, we favor a proactive setting up of the default forwarding ('match-action') rules by the SDN controller rather than reacting to the first packet of newly recognized flows. We also proactively set up the different alternate paths defined by the service chain for a flow that would be chosen dynamically as a result of the processing at an NF.

4.3 Handling service chaining

In this section we focus on the characteristics and special requirements of different service chains. As mentioned earlier, a service chain is a sequence of services that a flow needs to pass before leaving the network. We discuss the details of our protocol messages in Section 4.4.

4.3.1 Static Chains

A service chain is *static* when the sequence of NFs in the service chain is defined in advance, and it is not dependent on the output of NFs. Otherwise, a service chain is dynamic. F1 and F2 in Figure 4.3 are examples of static service chain and F3 is an example of dynamic service chain. While static chains may at first seem trivial to handle, in fact they may still require flow tags to properly route flows if the implementation of the service chain includes a cycle, causing a switch to be visited more than once.

Unambiguous static flows

We first address the simplest case where the static flow is recognized unambiguously at an SDNFV switch. This occurs when the flow just traverses this switch once, or

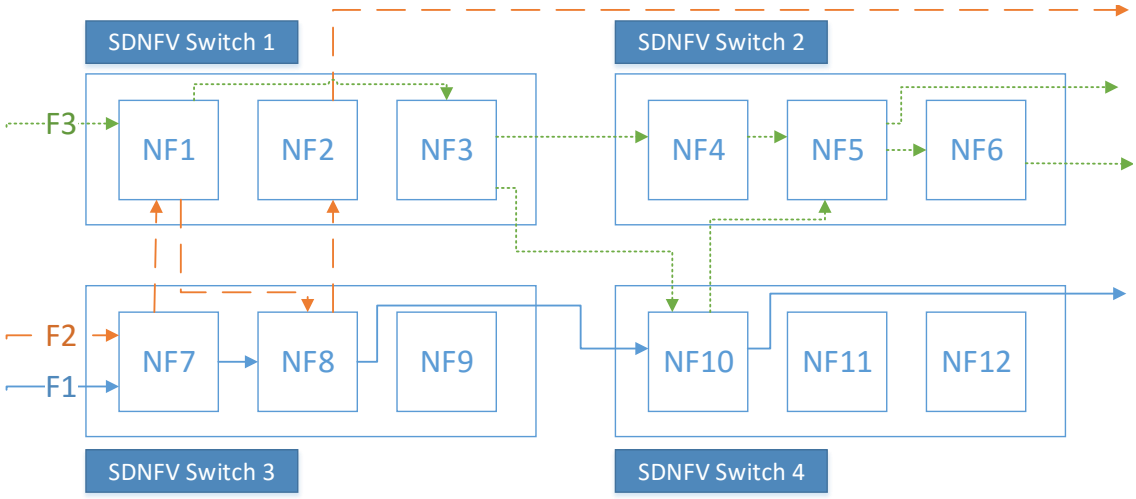


Figure 4.3: A set of three service chains traversing NFs at four SDNFV Switches.

if it is traversed multiple times, each visit is distinguishable from other visits. For example the input interface may be different between different visits. In this case, the forwarding decision can be made using available information and without any need for additional tags. Note that this applies even if a flow has to be processed consecutively at multiple NFs that reside on the same SDNFV Switch as a virtual port can be leveraged to recognize the flow's state. This is shown in Figure 4.3 where flow $F1$ traverses three different NFs on two different SDNFV Switch hosts; on each host the NF Manager enforces the correct NF processing order.

Ambiguous static flows

When a flow has to visit the same SDNFV switch more than once and the input port is the same, there is a need to differentiate the arrivals so as to forward the packet to the NFs correctly. For example, if Flow $F2$ in Figure 4.3 does not have a tag (or some kind

of persistent state in the NF, which we avoid) there is no way to determine which NF is supposed to examine the packet next since its n-tuple will be identical both times it arrives at the host.

To solve this problem we use a flow tag carried in the packet to indicate the stage of processing. Rather than use a unique tag for each stage (as has been done in previous work [72]), we only generate a tag for the ambiguous hops in a path where they are needed, and we interpret the tags in conjunction with the flow identified by the n-tuple in the packet header and the switch input port. This reduces the number of tags required across all the flows in the network.

Tags that assist with routing across SDNFV Switches are added to packets by the NF Manager. When the SDN Controller configures the flow table rules in the SDNFV Switch, for example to make Flow $F2$ send packets from NF8 to NF2, it will include a tagging action as part of the flow table entry. Likewise, the flow rules on the switch receiving the packet will be configured to include the tag as part of the match criteria to determine processing. We call this tag a global tag or *GTag* since it is not local to a single host, but transmitted as part of the packet.

4.3.2 Dynamic Chains

Active NFs can produce different outputs for different packets. For example, a cache may produce a Hit or a Miss as outputs. Flows may need to go to different NFs based on a Hit or Miss. The chain containing these active NFs is called a *dynamic chain.*

In this paper, the outputs of active NFs are called *NFOutputs.* The NFOutput is written to the packet descriptor, a software data structure in the NFV platform, not

in the packet itself, since it is only used locally at that host. (An approach similar to E2 [70] can be used.) The NF Manager, using rules defined in the SDNFV Application and installed by the SDN Controller, knows how to interpret this output to select one of the possible next hops. This allows NFs to provide input on how packets should be routed, but leaves the SDNFV Application in charge of determining how those outputs are interpreted. It also reduces communication overheads by eliminating excessive and unnecessary calls to the SDN Controller.

Ambiguity in routing can arise, in a manner similar to static chains. We use the following solution. The SDN Controller is aware of all the possible paths of a flow. It creates a union of all the paths of this flow, and executes an algorithm similar to what was used for a static chain, on the resulting set. Thus, we can recognize the points where there is ambiguity for a flow for which a global tag is needed. For example, in a static chain, if a flow were going from Switch 1 to Switch 3 twice, we would consider it an ambiguity, needing a tag to resolve the ambiguity. With dynamic chains, even if some packets of a flow first go from Switch 1 to Switch 3 in Path 1, while other packets of that same flow go from Switch 1 to Switch 3 in Path 2, there is still a point of ambiguity needing global tags to resolve it.

NFs may affect the routing of flows in different ways. For example, they can affect the routing at an upstream or downstream node. Also they can affect the routing temporarily or permanently. The temporary impact on routing by the output of an NF is only on the current packet. Whilst an NF's output may affect the routing of a flow permanently such that all the packets of the flow follow a different route until another output from an NF is observed. Underlined edges in Figure 4.2 show the permanent edges

in those chains. Affecting the downstream means the route to subsequent NFs for the flow is altered. Upstream implies the routing to NFs upstream of the NF that produces the output is altered, for future packets of this flow. Dashed lines in Figure 4.2 indicates the upstream edges, while the other edges are downstream edges. Handling downstream and temporary actions is done in a manner similar to static chains. Here, we only focus on persistent and upstream actions.

Persistent Actions

In some cases, it is desirable for an NF to make a decision not only on a single packet, but on all the remaining packets in the flow. For example, an IDS NF may mark individual packets as being 'clean' using *temporary* actions, but when it detects something 'suspicious', it may wish to send all remaining traffic to a Deep Packet Inspection engine for further analysis. Our platform supports this through *persistent* actions, which allow an NF to produce an NFOutput that will be used by the NF Manager to adjust the flow table entry, causing all subsequent packets to bypass the NF which produced the tag and proceed directly to the next NF.

Upstream Actions

Sometimes it is necessary to change the routing upstream based on the output of an NF downstream. In this case, the NF Manager may have rules set by the SDN controller to forward information back to the SDN controller. The SDN controller uses this information to indicate that a prior NF in the chain should cancel its persistent action, or cause a new action to take place. For this, we extend the OpenFlow messaging framework

to enable the SDN controller to send the name of the NF to the upstream NF Manager, and include in it the identity of the flow (IP n-tuple) and the specification of the new action (which we describe in the next section). Upstream actions can work in two ways. First they can reverse the effect of persistent output like the "Clear" edge in Figure 4.2). They can also simply change the path from an upstream switch (e.g., "remove policy" edge in Figure 4.2).

Handling header changing NFs

There are circumstances where an NF may change the packet header (e.g., network address translation (NAT)), thus requiring additional information to ensure the flow is handled appropriately. However, unlike [72], we divide the service chain into multiple segments—the first is from the entry switch to the first NF that changes the header. The flow is routed through SDNFV switches normally using the n-tuple of the packet. For the segment of the service chain after the 'header changing NF', tags are added to the packet as necessary. Note that not all of the fields of a header might be changed by the NF. For example, a NAT does not change the destination IP address and port number for an outgoing packet (towards the public Internet). Thus, given that the tag can be associated with specific fields of a packet, we associate it with the destination IP and port. Thus, the same tag value can be re-used for different destination IP and port pairs, thus again resulting in a significant reduction in the number of tags required (especially as there is likely to be diversity in the destinations of the flows traversing the NAT). Our solution can support multiple service chain segments if needed, although we expect this to be a rare situation.

4.4 Protocol and interfaces

We now describe the interfaces and protocol primitives for information exchange among SDNFV's system components. We note that instead of making changes and adding new fields and matching rules to OpenFlow to support concepts such as GTag and NFOutput, it is possible to reuse unused fields in the OpenFlow protocol. This is possible because the number of unique GTags and the number of bits necessary for communicating the NFOutput needed in our proposed method is small compared to existing approaches.

4.4.1 Protocol between NF Manager and SDN controller

The OpenFlow protocol addresses a number of interactions between an SDN Controller and a network switch. These are also used in SDNFV switches for setting up flow tables. We now look at the functionality needed in the context of the SDNFV switch hosting NFs. The administrator, and thus the SDNFV Application, SDN Controller and NF Orchestrator know a priori the capabilities of the network nodes and links. We anticipate that most of the Flowtable rules would be set proactively and the necessary NFs for flows based on defined service chains have been instantiated by the orchestrator in coordination with the SDNFV application.

Unambigious static flow routing

When routing flows from a passive NF that is part of a static service chain, the SDNFV switches use the OpenFlow tables set up by the SDN controller with the existing OpenFlow protocol primitives (e.g., based on a match-action rule using the IP 5-tuple)

through SDNFV switches as in a typical switch. This would be the case also when the flow is routed through a static path on the SDNFV switch, since the flow's route is unchanged. Note, we use a different logical port at the SDNFV switch to enable forwarding to the different local NFs.

Unambigious dynamic flow routing

With active NFs, we use NFOutputs to modify the route out of an SDNFV switch. The match-action rule would be enhanced by the SDN controller as:

$$\{\text{5-tuple, NFOutput, In port}\} \rightarrow \text{outPort}$$

Thus, the NF Manager includes the NFOutput produced by the NF and uses this to assist in selecting the next stage in the service chain.

Routing ambiguous flows

Global tags or GTags are used to enable NFs to influence the path of dynamic flows and to handle a flow arriving at the same SDNFV switch multiple times. In the latter case, when an SDNFV switch is shared among the different paths of a flow, a global tag is used to carry the current 'state' (visit) of the flow to enable the switch to forward the packet to the proper next hop. Rather than having a packet with a tag be forwarded to the controller as in [72], we depend on the SDNFV switch to make the routing decision using the tag. The forwarding table rule from the SDN controller to the NF Manager is enhanced as:

$$\{\text{5-tuple, NFOutput, GTag, In port, Push, Pop}\} \rightarrow \text{outPort}$$

GTag is the global tag. When an SDNFV switch has to insert a tag header as the packet leaves the switch, the Push flag is set in the flow table. When Push is 0, an SDNFV switch simply forwards based on the tag header. The tag header is removed by the NF Manager at a switch where the flow table entry has the Pop flag set to 1, possibly based on the GTag value. GTag is added to the packet before the switch where the ambiguous visits of the packets occur. For example if the packets of a flow go from Switch 1 to Switch 5 two times in different parts of their path, routing in switch 5 is ambiguous and GTag is added to the packet on Switch 1 and it is used at Switch 5 for routing the packet to next steps. GTag may be changed to another GTag later, based on need, to address further ambiguities. The GTag is removed by the last switch in the path.

Request for a change upstream

The output of an NF that changes routing upstream or decisions of upstream NFs has to be mediated by the SDN controller. The reversal of a decision at an upstream NF is communicated by the downstream NF (through its NF Manager) to the controller, to then be used to set the flow table entry at the upstream SDNFV switch as:

$$\{\text{5-tuple, NFOutput, GTag, In port, persistent}\} \rightarrow \text{outPort}$$

The flow is identified by the 5-tuple, potentially along with GTag. The NFOutput is used to communicate information to the local (downstream node's) NF Manager. The 'persistent' flag when set to 1 is used to indicate that the persistent decision at the upstream NF (as defined by the service chain) must be changed. Otherwise, it is a request for a normal change (without the tag) at the upstream NF.

Information for header-changing NFs

When an NF changes the header of the packet (e.g., by a NAT) and routing decisions at subsequent SDNFV switches have to be set by the controller, a global tag has to be added by the NF Manager at that switch. As discussed before, the global tag (GTag) is interpreted along with the unchanged parts of the packet header's 5-tuple, to effectively re-use the global tags. The controller sets the rules, as before:

$$\{\text{5-tuple, NFOutput, GTag, In port, Push, Pop}\} \rightarrow \text{outPort}$$

4.4.2 Communication between SDNFV Application, SDN Controller and NF orchestrator

Network status to placement engine

The information about available SDNFV switches, their capacity to run different NFs, the NFs currently running on switches and the link capacities, has to be communicated by the SDN controller to the SDNFV Application, and in particular to the placement engine. This will also include current performance information, such as load and traffic statistics to enable the placement engine to make incremental placement decisions, as described in [156].

Instantiation request to orchestrator

When the placement engine needs to create a new instance of an NF or the NF Manager wants to instantiate a new NF, the orchestrator needs to be involved. The request

to the orchestrator includes:

{SwitchID, NFName}

Orchestrating an NF Move

The SDN controller may initiate the move of an NF from one SDNFV switch to another for reasons of load balancing or to optimize the routing for a flow. For this the controller provides to the orchestrator:

{Src-SwitchID, Dest-SwitchID, NFName}

to cause the move of NFName from Src-SwitchID to Dest-SwitchID. The orchestrator and controller coordinate to perform a 'make-before-break' for re-routing the flow prior to removing the old NF instance.

Placement request for a flow

When a new flow arrives and the NF instances have not been created a priori, the SDN controller needs to make a placement request for the flow to the SDNFV Application (i.e., placement engine), with the flow information of the entry switch, the exit switch, and the assigned service chain. Upon creation of the NF instances by the NF orchestrator in coordination with the placement engine, the SDN controller updates the corresponding NF Managers with the flow table entries.

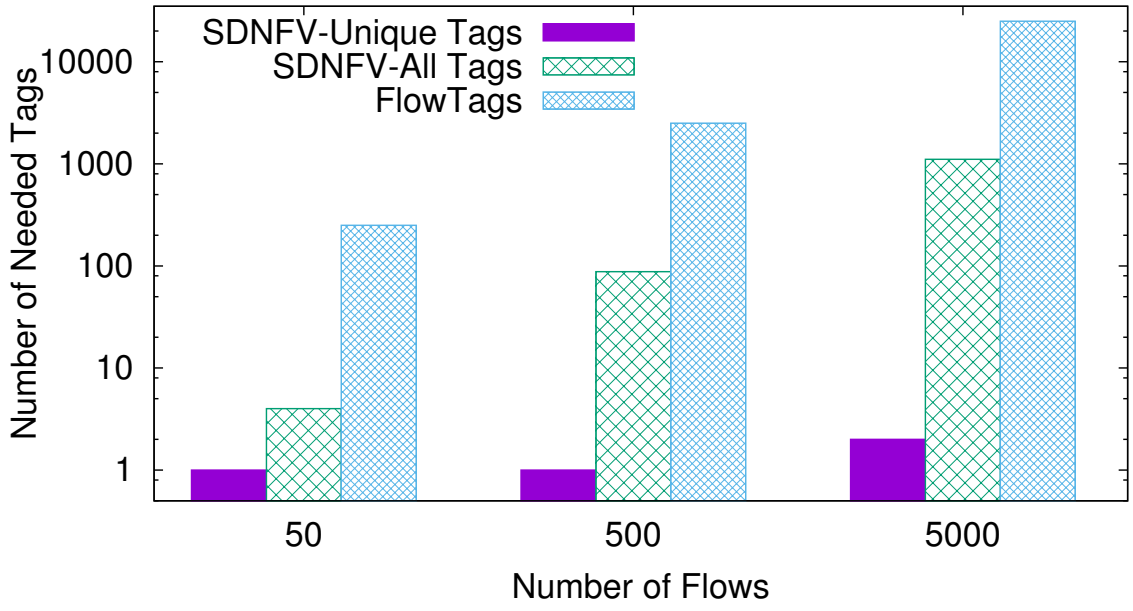


Figure 4.4: Number of tags necessary for managing varying numbers of flows

4.5 Evaluation

We evaluate the protocol framework for the SDNFV architecture proposed here along two dimensions: the number of tags necessary network-wide and the number of messages exchanged with the SDN controller. The first is important to ensure that we use the limited number of available bits in packet headers wisely, by effectively using the global tags when it is necessary. The second is to reduce the load on the SDN controller as much as possible. Both of these seek to make our architecture scale better.

The topology for our evaluation is from Rocketfuel [164] (AS-16631) with 22 nodes and 64 links. All the flows go through a service chain of length five and start at a random node in the topology and exit at another random node. Each SDNFV switch is able to

support two NFs and the placement and routing are decided by our placement engine [156]. We use a Java-based simulator to estimate the number of global tags needed.

4.5.1 Network with only NFs that don't alter headers

We first look at a network that only has NFs that do not alter the packet headers. The global tags then are used only for clearing ambiguities in the routing of the flows. Then we compare the number of tags needed in our approach with described method in FlowTags [72]. The result is shown in Figure 4.4. Since FlowTags needs a new tag for each flow on an NF, it needs 25000 tags for managing 5000 flows in this network when we have a static service chain of length 5. However in our approach, the number of tags needed depends on the paths of the flows. A tag is used only if an ambiguity exists in routing of the flow. As a first step, this itself reduces the number of tags needed. Furthermore, the number of unique tags needed can be reduced significantly by re-using the tag for different flows. For example if 10 flows need 2 tags each, the total number of necessary tags is 20, however we can use the same tags for all these 10 flows. Hence, the number of necessary unique tags is 2. The decision to route a flow then is based on the combination of the packet header's 5-tuple *and* the global tag. The consequence is that the number of unique tags is much smaller for SDNFV-Unique Tags, as shown in the Figure 4.4.

4.5.2 Network with NFs altering headers

For this case, the third NF in the service chain alters the packet header. We did two experiments. In the first one, the NF does not change the destination address. This is therefore leveraged to further reduce the number of tags necessary. In the second case,

No. of Flows	SDNFV				FlowTags
	All tags		Unique Tags		
	Dst.	Dst.+Ports	Dst.	Dst.+Ports	
50	7	2	1	1	250
500	285	8	1	1	2500
5000	4681	65	3	2	25000

Table 4.1: Number of tags in network with NFs altering headers

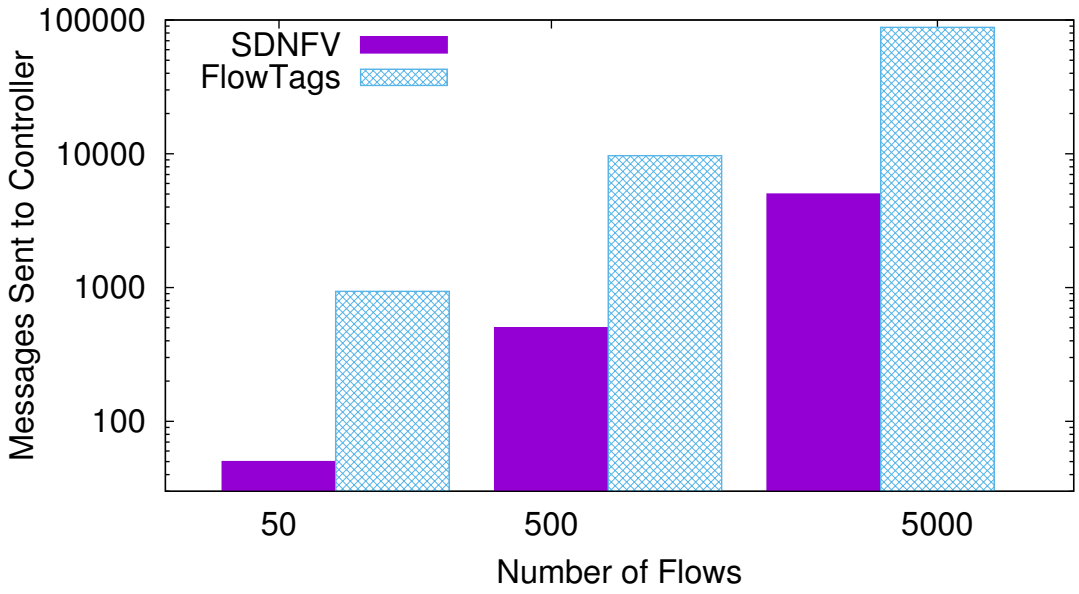


Figure 4.5: Number of messages sent to controller for SDNFV and FlowTags

the destination and the destination port are unaltered. In our experiment, the port number was uniformly distributed across 56K port numbers. The dramatic reduction in the number of tags needed, to just 2 or 3, is shown in Table 4.1. Finally, an important criterion for scalability is the number of messages sent to SDN controller. Our approach and FlowTags are compared in Figure 4.5. In SDNFV, each flow needs to contact the SDN controller once. However, with FlowTags each switch contacts the SDN controller to first get the necessary

rules for that FlowTag. Further, each middlebox contacts the SDN controller twice, first to generate the tag and secondly to consume the tags for a flow. SDNFV's hierarchical control dramatically reduces the overhead on the controller.

4.6 Related Work

Recently there has been work to increase packet performance on COTS servers [98, 143]. NetVM[98] uses zero-copy transfer packets from/to VM and between VMs at wire-speed. ClickOS [143] maps packets buffers to the VM's address space to reduce overhead for achieving high performance packet forwarding.

While there has been a large body of work on the core SDN protocols, we focus here on the work that examines SDN-control of middleboxes and network functions, especially for the dynamic use of middlebox functionality in the network.

SIMPLE[185] is an SDN-based policy enforcement framework for steering the traffic of middleboxes. The primary goal is to support complex routing to have flows traverse middleboxes. SIMPLE uses tags in a packet assigned to recognize a routing loop. However, SIMPLE does not address the need to support dynamic service chains and the necessary routing through them, especially when the service functions for a flow (or a class of flows) has to be updated based on the processing at a network (middlebox) function. The use of flow tags minimizes the size of the flow tables, which is a desirable characteristic we exploit in this work as well. However, our view is that tags can serve a much wider purpose of NFs communicating to the controller or to downstream NFs the result of their processing

so as to exploit the flexibility that is provided in our framework to support dynamic service chains.

Steering [243] is an SDN-based framework for dynamic routing of traffic through middleboxes. While it supports changing the service chains based on the output of the middleboxes, such an action is always mediated through the SDN controller which updates different tables in different switches. The framework is much simpler in that complex routing (e.g., loops) is not supported. Further, there is no support to dynamically place multiple replicas of an NF in a large network.

FlowTags [72] is an architecture for dynamically managing flows through middleboxes in the network. It also uses tags for complex, stateful routing of flows. A new tag is utilized for each branch of a service chain. Tags are not re-used across flows, hence they should be unique unless the flows are temporally or geographically separated. Packets of flows marked with tags are routed only based on the tags. A middlebox contacts the SDN controller to create the tag and for interpreting and consuming the tag. When the service chains are long and middleboxes have a large number of possible outputs, the number of tags required can be very large. In contrast, the autonomy we provide with our hierarchical control at the NF and NF manager levels avoids overloading the SDN controller and is thus more efficient. Moreover, by combining the tag and the full n-tuple in the flow table along with the input port and SwitchID for routing, our approach requires far fewer tags.

Slick [13] is a network programming architecture that provides the opportunity for the SDN controller and middleboxes to communicate with each other. The main feature of interest to us here is that it provide triggers for network functions to contact the SDN

controller. However, the framework continues to retain the 'master-slave' relationship between the network data plane and the SDN controller, unlike our approach of providing a hierarchy of control across the components.

Besides the aforementioned works which are done in SDN field, some other works such as Network Service Header (NSH) [68] of IETF are done in other fields too. However because of lacking a centralized controller, making a global optimal decision on different aspects such as assigning flows to different instances of NFs is not possible. Moreover, the dynamicity introduced in this work is limited in comparison to our selected approach. In our approach, the flows path is completely based on the dynamic output of the NFs and even an output of an NF may change the path of next packets in upstream. In NSH classifiers are separated from network functions and they cannot affect the next packets in upstream. The last difference that we want to point out is that NSH adds a variable size header to the packet, whereas in our approach because of a limited number of bits needed for defined fields, we are able to reuse the existing unused fields in other protocols.

4.7 Summary

Our SDNFV architecture seeks to achieve our vision of a dynamic and flexible network with a smarter data plane. The protocols we develop also enable NFs to update a flow's forwarding rules dynamically, subject to constraints specified by the service graph, without burdening the centralized SDN controller. This allows SDNFV to base flow management decisions on characteristics that cannot be determined at flow startup. It allows changing traffic characteristics across multiple flows to affect routing behavior, for example

by detecting DDoS attacks or other anomalous flows, or network policy and load-dependent modifications. Our architecture supports a rich set of NF types that dynamically modify the path of flows both upstream and downstream and ones that change packet headers, such as NATs. SDNFV enables dynamic instantiation of NFs by an orchestrator. The SDNFV architecture uses tags for NFs to communicate their output and the state of the flow by leveraging the idea of "FlowTags" but use it in conjunction with the knowledge of the network at the SDN controller, to dramatically reduce the number of tags used network-wide. We substantially reduce the overhead on the SDN controller by taking advantage of the hierarchical control possible with the smarter data plane. The smarter data plane includes the NFs, the NF Manager and ultimately, the SDN controller, in decision making.

Acknowledgment

This work was supported in part by NSF grants CNS-1422362 and CNS-1522546.

Chapter 5

Considerations for Re-Designing the Cellular Infrastructure Exploiting Software-Based Networks

As demand for wireless mobile connectivity continues to explode, cellular network infrastructure capacity requirements continue to grow. While 5G tries to address capacity requirements at the radio layer, the load on the cellular core network infrastructure (called Enhanced Packet Core (EPC)) stresses the network infrastructure. Our work examines the architecture, protocols of current cellular infrastructures and the workload on the EPC. We study the challenges in dimensioning capacity and review the design alternatives to support the significant scale up desired, even for the near future.

We breakdown the workload on the network infrastructure into its components-signaling event transactions; database or lookup transactions and packet processing. We quantitatively show the control plane and data plane load on the various components of the EPC and estimate how future 5G cellular network workloads will scale. This analysis helps us to understand the scalability challenges for future 5G EPC network components. Other efforts to scale the 5G cellular network take a system view where the control plane is separated from the data path and is terminated on a centralized SDN controller. The SDN controller configures the data path on a widely distributed switching infrastructure. Our analysis of the workload informs us on the feasibility of various design alternatives and motivates our efforts to develop our clean-slate approach, called CleanG.

5.1 Introduction

There are two new drivers that are likely to be of concern to cellular operators. The first is the shift to small cells, as a way to increase system capacity. If this is designed by retaining the current architecture and protocols, it is likely to cause a dramatic increase in the amount of hand-offs from user and device mobility that will place a considerable load on the control plane of the cellular network. There will also be significant signaling load from lower power devices, shutting down or going to idle states and connecting periodically as predicated by the services/applications running on them. Secondly, with new use cases such as Internet of Things (IoT) and Machine to Machine (M2M) communication, cellular traffic is likely to grow significantly, with the traffic mix and communication patterns expected to change significantly. In a lot of these new use cases a large number of control messages

and control events are generated for transferring small chunks of data, unlike traditional applications.

This shift in the control plane load is of concern to service providers. In the past, the provisioning of the cellular network control plane was based on traditional voice applications where the control plane load (based on call arrivals) was much smaller. For example, in the current cellular protocol design, there are over 15 control plane messages exchanged between the cellular end-device (user equipment (UE)) and the Evolved Packet Core (EPC) every time a UE transitions from idle to active state, and slightly over 30 control plane messages when a UE moves from being associated with one base station to another. With typical IoT traffic loads, service providers will need to scale their control plane capacity significantly to be able to handle a much larger number of control plane events per second.

A number of efforts to scale the 5G cellular network take a system view where the control plane is separated from the data path. To help understand the feasibility of different design alternatives, we carefully study the architecture and protocol processing of current cellular infrastructure, with particular focus on the workload on the EPC. We estimate control plane and data plane load on the various components of the EPC with the current 3GPP architecture for a representative instantiation of the EPC in the United States (basing the topology and design based on the aggregate traffic statistics seen in the US cellular network).

Our analysis justifies a need to re-think how cellular networks are designed to support mobility, IoT and data traffic better, with lower latency, higher throughput, and

higher overall system capacity. Current cellular networks have separate purpose-built hardware appliances for the Evolved Packet Core (EPC). A newer trend has been to build software-based platforms. Software-based platforms have also exploited virtualization that allows dynamic scaling. Virtualization not only enables scalability, but offers flexibility to dynamically move resources from one function to another. We show that a software EPC offers much needed flexibility and evolvability however, supporting a high volume of control traffic in conjunction with the data plane still continues to be a challenge.

To address these issues, we are currently working on an approach called CleanG, where we propose to simplify the components and the control protocol while keeping all the essential functionality supported in the current cellular network. By simplifying the protocol and taking advantage of a virtualization-based EPC, we can substantially scale up network capacity.

5.2 Current 3GPP Architecture

Figure 5.1 details a typical end to end wireless infrastructure. Traffic to and from devices or User Equipment (UE) connecting through a cellular base station (eNB) is aggregated and carried over backhaul links into the wireless core infrastructure, also called the Evolved Packet Core (EPC). The EPC core mainly comprises the MME, SGW, PGW, Home Subscription Server (HSS), Policy Control and Rules Function (PCRF) and the Policy Control Enforcement Function (PCEF). Each of these functions is generally instantiated on dedicated hardware racks distributed across the network deployment geography. Dis-

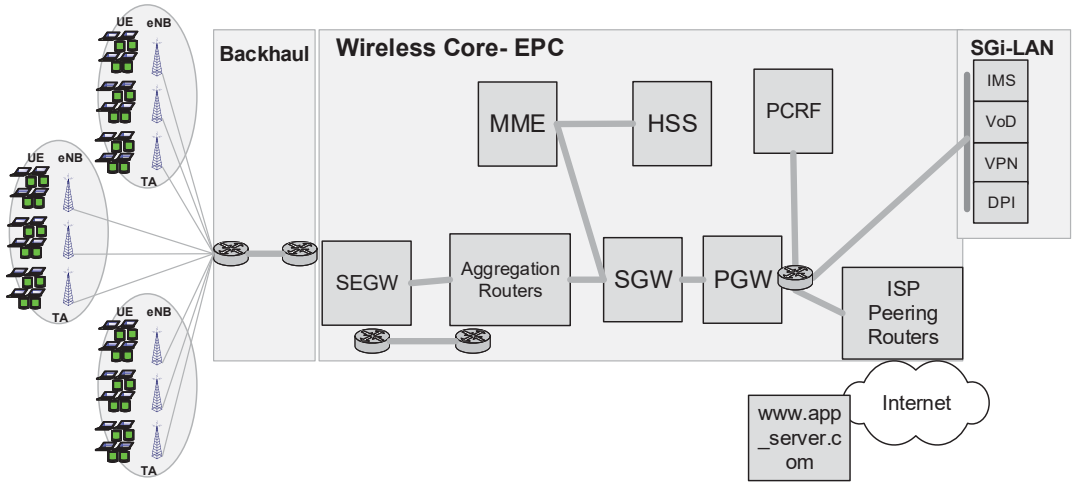


Figure 5.1: Typical end to end wireless infrastructure.(SEGW: Serving Edge Gateway MME: Mobility Management Entity HSS: Home Subscription Server SGW: Serving Gateway PGW: Packet Gateway PCRF: Policy Controls and Rules Function)

tribution of these hardware racks implementing the different functions is based on system capacity utilization and traffic engineering principles.

We now examine how traffic is handled. Traffic to and from the UE is filtered through templates (TFT), classified into flows identified by Quality Class Identifiers (QCI) and then metered through routers peering onto the Internet Backbone. Traffic metering is per maximum bit rate (MBR) value defined by the QoS associated with the flow. In addition to this, application- Application Detection Control (ADC) or carrier specific- Policy Control and Charging (PCC) rules based on flow identifiers may be applied to administer traffic into rate groups which can then be charged per billing policies.

The TFT, QCI, QoS prioritization, MBR, ADC, PCC flow characteristics need to be negotiated and defined before the flow can be established, i.e., before any data can

be sent or received by the UE. Therefore for a successful connection to be opened by an application on the UE, the signaling stack on the UE will need to go through a series of message exchanges with the EPC to negotiate and establish the characteristics of the flow that an application on the UE is attempting. For e.g., consider an Android application making a Http URL Connection call. For this call to return successfully, the signaling stack should have already negotiated the flow characteristic with the EPC.

A significant protocol overhead is the current architecture is in establishing the flow context in the cellular network, which we detail now. The end to end connection is identified by the typical 5-tuple- src_IP, src_Port, dst_IP, dst_Port and a defined transport protocol, TCP in the example above. The src_IP, src_Port is a UE IP address and Port on the UE IP stack and dst_IP, dst_Port is the remote service IP address and Port. The UE IP address is provided to the UE by the EPC core through a create session message sequence exchanged with the core. Another application on the UE e.g., VoLTE or an email client will need to establish a connection on a different Port. This connection context, uniquely identified by the 5 tuple, is maintained end to end between the application on the UE and the service hosted on the Internet and is carried in the IP Protocol Data Unit (IP PDU) header. It is mirrored in the Packet Data Convergence Protocol (PDCP) between the UE and the eNB. This is encapsulated into a tunnel header with end points being established and maintained between the eNB and the EPC core as long as the application on the UE is active. Flow characteristics (TFT, QCI, QoS prioritization, MBR, ADC, PCC) are communicated to the tunnel end points for appropriate treatment of the application context at the EPC core.

Each application session will need the creation of the tunnel and associated context or flow characteristics between the eNB and the EPC core. The establishment of the tunnels is negotiated through the MME in the EPC core by eNB. The flow characteristics are negotiated between the EPC core elements internally. At the time of flow creation or modification by the UE, the eNB signals the MME which then propagates these messages for tunnel establishment to the SGW after verifying the subscription policies for the UE with the Home Subscription Server (HSS). The SGW forwards the flow characteristic requests to the PGW. The PGW, in turn, queries the Policy Control and Rules Function (PCRF) and sets the internal Policy Control Enforcement Function (PCEF). The PGW then sends the UE IP address to SGW to be used for the session. The UE IP address finds its way back to the UE through the MME and the eNB. At this point, UE IP stack will have the UE IP address for the application to open a connection. The eNB and SGW will have the tunnel end point identifiers (TEID) to carry the traffic flow for the application connection. The PGW will have the TFT, QCI, QoS prioritization, MBR characteristics to match against the flow and enforce the ADC and PCC rules. Refer Table IV [66].

All of the control protocol/signaling overhead above is primarily between the eNB and the EPC core. The flow characteristics for the IP PDU are extracted to be applied on a bearer that carries the traffic for the session. In this manner, each session can have many bearers, the equivalent of many ports that can be opened on a single IP address by an application on the UE. Each bearer of a session has a tunnel established between the eNB and the SGW. Each session will need to have in addition an IP address provisioned by the PGW. According to the 3GPP specifications [154], each UE can establish up to 11

bearers. These bearers can all belong to a single session or can each be associated with a different session. In such a case, the UE will have been assigned up to 11 IP addresses.

In the Internet backbone beyond the EPC, QoS treatment of traffic is based on fields in the IP header. The justification for the overhead of extracting flow characteristics in the cellular network was done with few key intentions: a) the path for the flow was signaled ahead of time so an appropriate resource reservation could be made. This is particularly true for management of precious (limited) radio resources between the UE and the eNB. Further, resources need to be provisioned in the EPC core to ensure compliance with Service Level Agreements (SLAs). Resources to be reserved for a flow include switching, routing bandwidth, memory for storing flow characteristics, CPU processing capacity, memory and IO bandwidth for handling the packet flow; b) Correspondingly, signalling releases resources when not required so they could be assigned to another flow, to optimize capacity utilization, c) the IP PDU itself is not touched, instead the packets are enveloped and carried over the EPC core for distribution to the Internet end points.

The cellular distribution of the eNB, the mobility of the UEs across these cell boundaries, transient activity of applications on the UE driven by battery life considerations, require connections to be constantly torn down, re-established or activated. This means that the creation of the tunnels and flow characteristics associated to the application session context will have to be re-set or refreshed multiple times between the eNB and the EPC core re-triggering the internal negotiations among the EPC core functions- MME, HSS, SGW, PGW, and PCRF.

5.3 Overview of SDN and NFV

Software Defined Networking (SDN) is changing networks, particularly data centers, by providing a logically centralized control plane capable of using carefully designed rules that are specific to each individual flow[74]. SDN separates the networks control plane from the data plane, with a goal to allow flexible flow management by a logically centralized SDN controller. The control plane, implemented in the SDN controller, has complete authority in the determination of the path for individual flows, in conjunction with a minimal network data plane. SDN's separation of the control plane is an effective way for determining the actions to be performed on a flow when a network switch sees packets of a flow, by having the controller set up the match/action rules for the switch to use in forwarding packets. SDN applications perform traffic engineering, QoS, monitoring, and routing, and provide inputs to the SDN controller (through its North-bound interface(s)). The SDN controller then signals network elements to configure flow tables via standardized protocols such as OpenFlow [170] (via its South-bound interface(s)) based on the first packet or rules that are defined ahead of time. The data plane refers to these flow tables and sends packets according to the specified actions for matched rules. As we detail in Section 5.5, we outline several efforts in research and industry that seek to exploit this philosophy of SDN in re-designing the control plane for cellular networks. These efforts seek to make the SDN controller responsible for setting up the path, managing resources and addresses for UE sessions.

Another technology direction is the development of Network Function Virtualization (NFV), a technique to run high performance network services as software running in

virtual machines (VMs) on commodity servers [98, 143]. By using commercial off-the-shelf (COTS) servers and virtualization technologies, NFV provides network and cloud operators greater flexibility for using network-resident functions at lower cost (riding the lower cost vs. performance curve of standardized high-volume servers). Using a virtualized architecture greatly improves the ease with which new network functions (NFs) can be deployed; each is given its own isolated virtual environment (be it VMs, or the newer approach of using Containers), providing performance isolation and simplifying resource management. Services residing in NFs can be easy to develop user-space software applications.

Together, SDN and NFV promise to increase the flexibility of networks at both the control and data plane levels. In this paper, we carefully examine the benefits and limitations provided by these technologies in meeting the challenges posed by the imminent changes and scaling up of the load for cellular networks. It is important to keep in mind that the control plane in the cellular network needs to be much more tightly coupled with the data plane because of the limited resources in the air-interface, requiring an admission-controlled handling of individual flows. This is another motivation to examine carefully, the applicability of SDN and NFV for virtualized EPC environment, based on a quantitative understanding of the workloads seen in current cellular networks and projecting a scale up of these workloads for future 5G networks. We observe below that unlike a typical data center (or even IP network) where there are a large number of data packets for a flow, the number of data packets that are processed for each control event in a cellular network is relatively small. In an IP network, the SDN controller is involved in a control processing event (e.g., at the beginning of the flow) that is amortized across a large number

of data packets for that flow. Thus, the load on the centralized SDN controller is more manageable. However, as we observe in the section below, the cellular network sees a much larger number of control events (i.e., transactions) for each UE, with a relatively small ratio of the number of data packets for each control event. Moreover, there is a tight coupling, as the data flow may block the completion of the processing of the control event. It is for this reason that a straightforward implementation of the control plane on an SDN controller is difficult to scale up. While we state this as intuition here, we provide a detailed quantitative justification that validates this determination regarding the right partitioning between the control plane and the data plane in cellular networks. This is the motivation for our current work on a clean-slate re-architecting of the cellular infrastructure with our proposed CleanG architecture.

5.4 Cellular deployment topology, workloads, system impact and 5G implications

To understand the implications of control signaling protocol overhead on the EPC core, we look at a typical nationwide deployment first, with the full understanding that these are estimates (as individual cellular operator infrastructure characteristics are highly proprietary). We then break down the dimensions of the workload and infer the stress vectors at the EPC core.

Consider the United States: we estimate the number of UEs to be 400M, rounding off to having 1 UE per every person in the country. Based on the current cellular

EPC core workload dimension	**Stress Vector**	**System Impact**	**Comments**
Control Plane	Signaling terminations: 2000 UE : 1 eNB 1340 eNB : 1 MME | 3 MME : 1 SGW; 1 SGW : 1 PGW;	2.7M UE : 1 MME| 8M UE : 1 SGW; 8M UE : 1 PGW;	n : m relationships between EPC core and eNB infrastructure and intra EPC core functional elements. 200K eNB : 150 MME :: 1340 eNB : 1 MME 2000 UE/eNB x 1340 eNB/MME = 2.7M UE/MME 200K eNB : 50 SGW :: 4K eNB : 1 SGW 2000 UE/eNB x 4K eNB/SGW = 8M UE/SGW
	Signaling event or Transactions Per Sec (TPS)	100K TPS : 1 MME 100K TPS : 1 SGW	10 uSec/Transaction
User Plane	User Plane terminations: 2000 UE : 1 eNB 4K eNB : 1 SGW; 4K eNB : 1 PGW;	8M UE : 1 SGW; 8M UE : 1 PGW;	p : q relationships between EPC core and eNB infrastructure and intra EPC core functional elements. 200K eNB : 50 SGW :: 4K eNB : 1 SGW 2000 UE/eNB x 4K eNB/SGW = 8M UE/SGW
	Packet Arrival Rate or Million Packets Per Second (MPPS)	31 MPPS : 1 SGW; 31 MPPS : 1 PGW;	33 nSec/IP PDU Or 1 Transaction every 304 IP PDU

Table 5.1: EPC core workload dimensions mapped to stress vectors and system impact

Dimension	Stress Vector	Impact
Control Plane	20M eNB : 50 SGW :: 400K eNB : 1 SGW	6.7M TPS : 1 SGW or 150 nSec/Transaction 80M UE : 1 PGW;
Data Plane	80M UE : 1 SGW	122 MPPS per SGW; i.e., 28Tbps/50 SGW/(575Byte*8)= 122 MPPS 9.0 nSec service time budget to process an IP PDU i.e., 1 control transaction every 17 PDUs

Table 5.2: 5G stress vectors

infrastructure deployed in the US, these 400M UEs connect to over 200K eNB [42] through roughly 50 EPC core infrastructures. This means that we will have at least 50 PGWs in the US across all the network operators. These assign UE IP addresses for establishing traffic flows to the Internet backbone. With a S-GW associated with each P-GW, and a ratio of a pool of 3 MMEs connecting to 1 SGW, we will have 50 PGWs, 50 SGW and 150 MMEs representing the deployed cellular network in the US.

To set up each application session's flow (e.g., an IP flow), there is signaling overhead for the eNB, MME and the S-GW and P-GW, along with message exchanges with the HSS and PCRF. Successful negotiation of the signaling establishes the bearer traffic between the eNB and the SGW. The data path goes through the S-GW, where the tunnel encapsulation and decapsulation are performed, while the P-GW sends and receives those data packets and in accordance with the PCRF rules forwards packets received to or from the Internet.

Thus, between the eNB and the EPC core there are two different planes in operation: A Control Plane that negotiates all the signaling ahead of the bearer traffic and a Data Plane or User Plane that is the actual IP PDU flow through the SGW, PGW and onto the Internet. The Control Plane and User Plane represent the dimensions of workload the EPC core needs to process.

We now examine the workload seen at the EPC, which drives the transaction processing capacity requirements for the hardware on which the MME, HSS, SGW, PGW and PCRF EPC core functions may run on. This transaction processing capacity, in turn, places requirements on the database and event or interrupt processing capacity needs from

the hardware. As observed in [66], the primary cause of limiting of EPC core capacity is the MME to S-GW interaction. Since the SGW sits at intersection of the Control Plane and the User Plane, interactions with the MME in establishing, re-activating or changing the traffic bearer tunnel end points, would directly impact the service time available for the bearer traffic (data path). Unless these interactions with the MME are properly separated they would starve the CPU, increasing its waiting time as it processes IP PDUs coming in from the bearer traffic (or delay the data path if the control path is provided priority).

Consider the above ratios: there are approximately 4K eNBs per S-GW, and 2000 UEs per eNB, we see that there are approximately 8M UEs associated with each S-GW (see Table 5.1. These ratios are driven by the lowest service time afforded by the chaining of SGW and PGW functions within the EPC core. The service time is the amount of time taken for processing a single IP PDU of the incident aggregate bearer traffic. This would be the time it takes the IP PDU to reach the CPU once it lands on the IO of the hardware implementing the SGW or PGW function. Even if the IP PDU lands directly on the cache of the CPU dedicated to the SGW or PGW function, any CPU waiting time spent by the CPU in fetching bearer tunnel end points, TFT, QCI, QoS prioritization, mbr, ADC, PCC flow characteristics for processing the packet, would adversely impact the service time.

Let us consider the data path first. We provide some conservative estimates of traffic seen at the S-P GW, and the available processing time budget for the data path. The traffic backhauled into the EPC core infrastructure from the 200K eNBs across the Unites States in 2012 was 7 Tbps (rounded up from 6.96 Tbps) [102] i.e., 140 Gbps bearer (data) traffic on average through each of the 50 S-P GWs. For an average IMIX packet size

[69] of 575Bytes, the packet arrival rate at each SGW and PGW of the EPC core in the US is 31 MPPS (= 140 Gbps/[575*8]). The service time budget for each IP PDU available at each of the S & P GW is therefore 33 nSec (= 1/31 MPPS).

Now, let us consider the control plane. Assuming that 50% of the UEs are active at any time (e.g., smartphones are ON all the time), of the 8M UEs associated with an S-GW, 4M UEs generate an event once a minute. Each event is assumed (typically) to require the flow characteristic of a bearer (flow) to be updated, resulting in 0.02 transactions per second (TPS) per UE to be handled at the S-GW, resulting in approximately 100K TPS at the S-GW across all 8M UEs (4M active) incident on it. Since the MME sees almost 3 times the interactions (S1 MME vs. S11 interactions) seen by an S-GW [66], the signaling incidence on the MME will be 3x or 0.05TPS/UE. We consider 50% of the UEs are active at any time in the calculations of Table 5.1.

A further analysis of the current control plane processing overheads in the S & P-GWs is in order to determine the processing time budget. As currently deployed, the signaling overhead in establishing/re-activating the bearer tunnel end points, extracting the TFT, QCI, QoS prioritization, MBR, ADC, PCC flow characteristics for processing the IP PDU is reflected on the SGW as a transaction every 10 μSec or every 304 packets. Each transaction would require the SGW function to ascertain the existence of the tunnel associated with the bearer and trigger the PGW to re-negotiate the TFT, QCI, QoS prioritization, MBR, ADC, PCC flow characteristics. The PGW will in turn accordingly update its PCEF with the flow characteristics. Given the 33 nSec service time budget imposed on the SGW and PGW, any locks on the flow characteristic resources during the transaction

update would block the bearer (data path) traffic. The CPU processing the IP PDU would be blocked and unless the packet backlog is cleared in the remaining 10 μSec transaction window time, the system will start to drop packets.

The aggregate Control Plane and User Plane load on each MME and SGW in the United States is tabulated under System Impact. This careful sizing of the control plane load for the EPC components of the cellular network is vital to determine the feasibility of the various architectural alternatives that would be considered even for migrating the *current* EPC to a software based environment, and the use of SDN for managing the cellular control plane.

Finally, let us speculate on the challenges we will likely face as we evolve to a 5G network with much larger numbers of end points, and frequent mobility due to small cells. With 5G, the number of UEs is expected to increase 10 fold, and eNBs expected to increase 100 fold (especially because of small cells) in the same deployment area [43]. Thus for 5G, we will have 4B UEs connecting through 20M eNBs. If we now conservatively assume a 10x increase in the signaling traffic from each UE and take the backhaul capacity at 28Tbps [102] the load on the SGW for processing each IP PDU is shown in Table 5.2. In 5G the signaling overhead would translate to 1 transaction every 17 IP PDUs(=122/6.7). This analysis, which we believe is a conservative reflection of the dramatic scale ups predicted in the industry, should indeed get us to think as to how to evolve the EPC to meet the imminent challenges we have to face.

5.5 Overview of Efforts to Re-architect 5G Cellular EPC

As we estimated, in the future the workload seen on 5G networks shrinks the service time budget of the SGW and PGW data plane functions for each IP PDU to about 9 nsec. More importantly, it reduces the number of data packets processed between signalling transactions to establish/re-activate the bearer tunnel end points of flows significantly. In fact, with 5G the S&P GW functions will experience an interrupt from a signalling event every 17 data packets With the service time for a data plane packet down to 9 nsec, the instruction budget is reduced by almost a factor of 4 i.e. from 180 instructions of current deployments to 50 instructions, assuming a CPU clock speed of 2.7 GhZ and a practical at best Cycles per Instruction (CPI) of 0.5. This tremendously complicates the design of hardware/software implementing the SGW and PGW functions. In a total of 153 (9*17) nsecs, the SGW has to complete processing 17 IP PDUs and the signalling transaction, so as to not fall behind. This has prompted a number of efforts to re-architect the cellular infrastructure, both in research and industry efforts. With the thrust to migrate to COTS hardware, system approaches to re-architect the cellular infrastructure for 5G, are being explored. Several of these seek to separate the control plane processing from the data path (conceptually similar to the direction taken by SDN in IP and datacenter networks). Thus, the efforts [36, 149, 27] primarily target at reducing the interaction between signalling and handling the bearer traffic. Another thrust being investigated is the re-distribution of the cellular infrastructure to reduce the ratio of the number of eNBs to the EPC core instances. Further, within that, reducing the ratio of the number of eNBs to the number of SGW instances is also being considered, to relieve the total load on the SGW (and

EPC in general). In line with this latter approach to reduce the ratio of the interacting infrastructure elements, an industry approach is to consolidate the MME and the signalling front of the EPC core while keeping the SGW and PGW that handles the bearer traffic integrated close to the eNB [65].

Let us now examine the use of an SDN-based approach for the 5G network infrastructure we mentioned above. The figure below details an approach gaining industry focus for eliminating direct signalling interactions with the individual components handling the bearer (data plane) traffic. An SDN Controller interfaces with the MME, HSS, and PCRF, centralizing establishment or re-activation of tunnels and flow characteristics. Variants of this approach merge the MME into the SDN controller, so the eNBs interface directly to the SDN controller. This approach, if successful in delivering the operational scaling i.e. workload implications of 5G, will bring the benefits of Data Center infrastructure to cellular networks [36]. Of course, the key to this approach succeeding is the ability to support the 5G workload requirements, especially the signalling transaction rate (in TPS).

Continuing our quantitative analysis of the signalling load (in a direction similar to the previous section), let us assume that the efforts to reduce the ratio of the eNB to the EPC core instances and number of UEs per eNB (with small cells) is also economically successful. With that we conservatively estimate that there are only about 250K UEs associated with each S&P GW (to keep the data plane load manageable on each), with about 40 S&P GW instances to support 10M UEs. The total number of instances of S-GWs nationwide (in the US) would be of the order of 16K to support the total UEs we might expect the 5G network to support in the US (4B UEs). Thus, we break down the

number of bearers handled by each SGW and PGW (S-PGW) instance to 250K bearers (data flows). This would bring down the packet arrival rate to 1 MPPS increasing service time budget per IP PDU to a more manageable 1 μsec. The control plane transactions on each SGW instance at 5G workloads will reduce to about 100K TPS or an inter-arrival time of 10 μsec, approximately 10 data packets for every signalling control event. This means having 10 μsecs to process 10 IP data PDUs and 1 control plane transaction at each S&P GW. Compared with the 153 nsecs computed earlier, this is an order of magnitude larger service time budget for the S & P GW functions. This should allow enough processing time for IP PDUs between control plane transactions on COTS hardware S&P GWs.

The analysis above has significant implications on an SDN-based approach for supporting the control plane load. A control plane load of 100K transactions per second with 5G workloads would mean that an SDN controller will have to sustain 100K TPS on the south bound interface to an individual S&P GW (note that there are about 16K SGWs nationwide, potentially in clusters of 40 S&P GWs, each supporting 10M UEs per cluster). With an SDN Controller to S&P GW ratio of 1 : 40, the north bound interface of the SDN controller will also have to handle close to 400K TPS to sustain all 40 S&P GW functions. Note that each of these transactions may have to be handled distinctly, to avoid the latency penalty that comes from batching these transactions to do a single aggregate update. Such batching delays the processing of control transactions. The resulting lag can impact data plane performance. Packets may be forwarded based on outdated state information (e.g., after device mobility), resulting in wasted resources and packet loss. These performance requirements pose challenges to current SDN controller architectures and implementations.

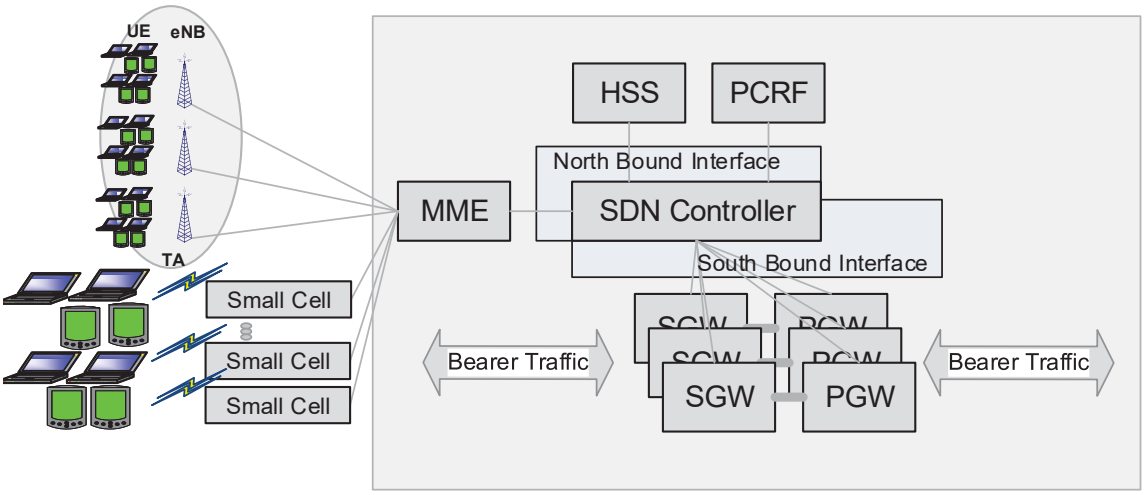


Figure 5.2: 5G SDN based Network Infrastructure

5.6 Discussion

With the tremendous growth anticipated in cellular network control plane workloads (with IoT traffic and the deployment of small cells), the EPC components in the network need to scale up significantly. Using representative workloads we expect on a typical EPC in the US, we quantitatively show that in future 5G networks the control plane load incident directly on each S&P GW function would be over 100K TPS with over 8M signalling TPS arriving at the EPC core. Current approaches considering the use of a centralized SDN controller will need to have the capacity to handle 400K TPS and transform these to 100K TPS to manage flows on each of the associated 40 S&P GW functions. To address these challenges, our current efforts on CleanG seek to address these challenges by examining both a new architecture for consolidating the EPC components, and a considerably simplified control protocol to ease the overhead, thus enabling scalability and achieving low latency.

Acknowledgment

This work was supported in part by the US National Science Foundation grant CNS-1522546.

Chapter 6

CleanG: A Clean-Slate EPC Architecture and Control Plane Protocol for Next Generation Cellular Networks

Cellular networks play a dominant role in how we communicate. But, the current cellular architecture and protocols are overly complex. The 'control plane' protocol includes setting up explicit tunnels for every session and exchanging a large number of packets among the different entities (mobile device, base station, the packet gateways and mobility management) to ensure state is exchanged in a consistent manner. This limits scalability. As we evolve to having to support an increasing number of users, cell-sites (e.g., 5G) and the consequent mobility, and the incoming wave of IoT devices, a re-thinking of the

architecture and control protocols is required. In this work we propose CleanG, a simplified software-based architecture for the Mobile Core Network (MCN) and a simplified control protocol for cellular networks. Network Function Virtualization enables dynamic management of capacity in the cloud to support the MCN of future cellular networks. We develop a simplified protocol that substantially reduces the number of control messages exchanged to support the various events, while retaining the current functionality expected from the network. CleanG, we believe will scale better and have lower latency.

6.1 Introduction

There are two new drivers that are likely to be of concern to cellular operators. The first comes from the shift to small cells being undertaken as a way to increase capacity. This is likely to cause a dramatic increase in the amount of hand-offs caused by device mobility, placing a considerable load on the cellular control plane. Secondly, with new use cases such as Internet of Things (IoT) and Machine to Machine (M2M) communication, cellular traffic is likely to grow significantly. The workload change because these new use cases will result in a larger number of control messages generated. Along with the workload change comes a need to increase data throughput as well as significantly reduce latency.

Current cellular networks carry data over virtual tunnels that require considerable protocol overhead for setup. Driven by software and hardware limitations of earlier platforms, the control plane elements for handling mobility and device state management (the Mobility Management Entity (MME)) has been separated from the data plane (S&P gateways). This distribution of functionality among a set of distributed components results

in significant protocol overhead. Our recent paper [158] describes the challenges for the current 3GPP architecture and protocols [86] to meet the performance requirements in a typical nationwide deployment. A newer trend has been to build software-based platforms, especially for the EPC which includes the MME and S&P GWs. While a software EPC offers much needed flexibility and evolvability, supporting a high volume of control traffic in conjunction with the data plane still continues to be a challenge. As this change is occurring, however, it is important to re-think the protocols (especially control protocols) used in cellular networks, not just re-implement the same set of protocols in software.

Consider for example the current cellular protocol, as specified in the 3GPP specifications [86]. One aspect we focus on with the protocol is the need to set up an explicit GPRS Tunneling Protocol (GTP) Tunnel that has to be set up for each session between the base station (eNB in LTE networks) and the S-Gateway. The MME mediates the set up of the tunnel. The protocol has to ensure the state is coordinated and consistent across all of these distributed entities, ensuring that the tunnel ID and state are set up at the eNB and S&P GWs prior to data flow. Data flow over a simpler tunneling approach such as GRE [84] or VxLAN [219] has the potential to considerably simplify the control plane for this function. Similarly, there are a number of additional overheads which a revised, clean-slate approach could simplify. Overall, in the current cellular protocol design, there are about 13 control plane messages exchanged between the cellular end-device (user equipment (UE)) and the Evolved Packet Core (EPC) every time a UE transitions from idle to active state, and slightly over 20 control plane messages when a UE moves from being associated with one base station to another. This is unsustainable with the anticipated

workloads in a 5G network, as we described in [158]. We believe there is a critical need to re-examine the control protocols in the light of current usage (much more data than circuit switched voice) and expected future evolving usage (IoT, small-cells) and workload. We recognize the need to retain several of the fundamental user requirements and the nature of an admission-controlled radio access network in a cellular network. User's expect a secure wireless channel, requiring authenticated and authorized access, when communicating over the cellular network. Our efforts with CleanG are focused on simplifying the control plane, without fundamentally changing the UE or eNB functionality of the cellular network. Within these constraints, we believe CleanG takes a new, clean-slate approach, in comparison to the many efforts currently being pursued in industry and in the literature.

Our CleanG approach exploits a virtualized software core. CleanG also seeks to consolidate the control plane functions of the S&P GWs and the MME. This eliminates a number of the control plane messages required to have a consistent update of the UE session context across the distributed components in the EPC. With the consolidation, the state for the UE can be retained in a single entity. Secondly, as we observe below with the CleanG protocol design, having the session context limited to just the two entities (eNB and the Clean-G Core), simpler tunneling protocols without an explicit setup can be adopted.

6.2 Data/Control Plane Load

This section describes a framework for how we size the processing load in the Data and Control Planes in the current 3GPP architecture. By examining the various elements of processing overhead, we arrive at a structured reasoning of the various architecture and

protocol alternatives for the cellular control plane. This provides a baseline for comparing architectural alternatives for capacity scaling.

6.2.1 Control and Data Plane Workload

In [158], we describe how traffic is handled in the cellular network, including the handling of QoS and policy. The MME is the signaling interface for the EPC core. Incoming signaling messages to the MME trigger a query waterfall to the HSS, SGW, PGW, and PCRF. Successful negotiation of the signaling establishes the bearer traffic between the eNB and the SGW.

Considering a typical United States-wide deployment with 5G, where the number of UEs and eNB are expected to increase 100 fold [43], we describe the anticipated workload for the current LTE network and for 5G in [158]. We note that the service time budget for each IP PDU is just 9.0 nSec, with a signaling transaction occurring once every 17 packets.

6.2.2 Implications of Signaling Transactions on Data Plane

Operations to look up the TFT, QCI, QoS prioritization, mbr, ADC, PCC flow characteristics, extracting the tunnel end points and applying to each IP PDU need to be accommodated within the available packet processing service time budget. if S_t is CPU operation time on each packet and SB_t is CPU stall cycles on each packet(service blocking time), service time budget is the time we have process each packet and is equal to:

$$(S_t + SB_t)/IP\_PDU$$

With Direct Data IO (DDIO) [112], CPU stall cycles on each packet is mainly composed of memory stall cycles (MS_cycles, measured in CPU clock cycles) on account of cache misses. Let T_s be the CPU clock cycle time. Let the cache miss penalty, measured in CPU clock cycles, be C_{mp}.

$$MS\_cycles = CacheMisses/Packet * C_{mp};$$
$$SB\_t = MS\_cycles * T_s; \text{That is:}$$
$$SB\_t = CacheMisses/Packet * C_{mp} * T_s;$$

Signalling transaction will invalidate the cached TFT, QCI, QoS prioritization, mbr, ADC, PCC flow characteristics, tunnel end points values, causing memory stall cycles. Let Tpp = Transactions/packet.

$$Tpp = CacheMisses/Packet * C_{mp} * T_s \tag{6.1}$$

Thus, the budget for the number of control transactions that we can support for each data packet is obtained in Equation 6.1. As we noted, as we move to support 5G workloads, there is a 3x reduction in Service time budget and 18x increase in the Transactions/packet or the Service Blocking time, compared to supporting current LTE workloads.

Our proposed CleanG protocol and architecture significantly reduce the Transactions/packet as well as the number of operations for packet processing. The sections that follow detail CleanG, comparing it with the traditional protocol and architecture. While the design is preliminary, we expect to demonstrate functional conformance of CleanG together with the potential saving in processing time per packet resulting in improved headroom on

service time budget with detailed implementation based performance measurements in our future work.

6.3 Other Efforts

Industry efforts to re-architect the cellular core have been focused on improving latency by bringing resources closer to end users. One of these is ETSI's Mobile Edge Computing (MEC) where computation and storage resources are provided in the macro eNB or at aggregations sites to provide low latency, high bandwidth, or having access to radio information. This work is orthogonal to our work, and CleanG could also benefit from the low latency the eNB as well as having access to radio information. We see this as part of our future extensions to the current CleanG design.

Other efforts focus on the disaggregation of the data and control components of the packet core. They depend on having control messages processed by an SDN controller. They also support a disaggregated and virtualized RAN. However, these design potentially impose additional delays resulting from the need to update the data path. It requires a number of additional steps for the SDN controller to update the data plane, which is not unlike the actions taken by the MME to update the variables in the data path for the S & P G/W. This is in contrast to the CleanG approach that leverages the common shared memory between the data and control plane components resulting from our consolidation of the EPC components. [67] investigates the performance of an SDN-based mobile core network. They observe that the delay between the controller and the switches significantly affects system performance. Our shared memory approach will considerably reduce this

delay. Next generation cellular networks will have tighter delay budget for completing control plane transactions (1 ms [90]), and the batching of switch flow-table updates, often exploited in SDN may not be exploitable. In a number of scenarios, such as initial attach and idle-to-active, the control plane delay directly affects overall user experience, and delay in the data plane. With handovers, user experience is affected as the switchover to the 'best' eNB would be delayed by a slow control plane.

While these industry efforts seek to achieve lower latency and better scalability, we believe they do not yet address the core problem because they continue to retain the complex 3GPP control plane protocol and an excessive amount of state (some quite unnecessary as we show in this paper) for each session.

6.4 Clean-G Architecture

The CleanG Architecture takes a clean-slate approach for the future (5G and beyond) cellular architecture. We re-examine the set of control protocols for CleanG, with the goal of achieving substantial simplification, by moving away from a virtual-circuit, tunnel-based communication between the eNB and mobile core network. Secondly, the protocol seeks to optimize the normal case, taking advantage of the QoS framework typical of IP based networks, and recognizing that the primary bottleneck is the air-interface. The protocol simplifications also reduces the amount of state maintained at the core for each session.

While we focus in this paper on a discussion of the CleanG protocol in the context of a particular instantiation of the CleanG architecture, our clean-slate CleanG protocol

could be applicable to many of the proposed alternative architectures for the 5G network that are being suggested in the industry. For example, the M-CORD approach could take advantage of the CleanG control protocol to both simplify the actions to be executed at the SDN controller as well as the amount of state information that has to be exchanged between the SDN controller and the data plane components. Similarly, the CleanG protocol could be applied in the context of the mobile edge cloud where the components are disaggregated. However, we believe that the most benefit will be with the CleanG virtualization-based architecture describe here.

The CleanG architecture is based on software EPCs built on virtualization platforms, enabling dynamic adaptation of the capacity for the data and control planes of the cellular architecture. CleanG seeks to exploit a large, scalable software based packet "CORE" that replaces the EPC elements of current cellular networks, and supports a large number of eNBs. In the CleanG architecture, the "CORE" consolidates the S&P GWs and the MME on the same host or cluster of hosts (potentially as separate functions implemented in distinct VMs or Docker Containers). This is in contrast to the current distributed implementation in cellular networks, where the functions are separated into distinct physical elements connected by network links. This consolidation of the EPC core functions enables a much more simple and lightweight control protocol as we describe below. The system design in CleanG seeks to cleanly separate the cellular control plane events from the data plane. By exploiting the capability of virtualization, we can instantiate additional instances of the core nodes on an as-needed basis and resources can be easily moved between the control plane and data plane components as the workload demands.

To assign processing resources (CPU cores) to the data and control planes on demand, we implement the CleanG Core as two sub components: Core-data, and Core-control. Core-data, is responsible for receiving packets, and either forwarding these packets to the appropriate downstream eNB, upstream network interface or direct control plane packets to the Core-control component. Internal state for each user session is maintained in memory structures shared between Core-data and Core-control, with the initialization being performed by a central controller/configuration management function that also takes policy and individual user information (accounting, authorization etc.) into account. Subsequent to completing a user 'attach', Core-control is primarily responsible for control plane transactions (e.g., idle-to-active, handover, etc.) at the CleanG EPC.

The CleanG core components are implemented as Network Functions (NFs) in the OpenNetVM[241] framework. The OpenNetVM NF Manager instantiates the Core-data and Core-control sub-components and dynamically allocates resources. The NF Manager interfaces with the NIC using the Intel DPDK libraries and manages the shared buffers between the NFs.

6.5 Control Protocols

While the move to a virtualized EPC (e.g., software based S & P-Gateways) helps address the scalability, it also offers an opportunity to re-think the control plane protocols used in cellular networks, not just re-implementation of the existing set of protocols in software. By embracing the scalability of the software platform, we can eliminate a significant amount of the control plane load arising from mobility, as the state for the UE can be

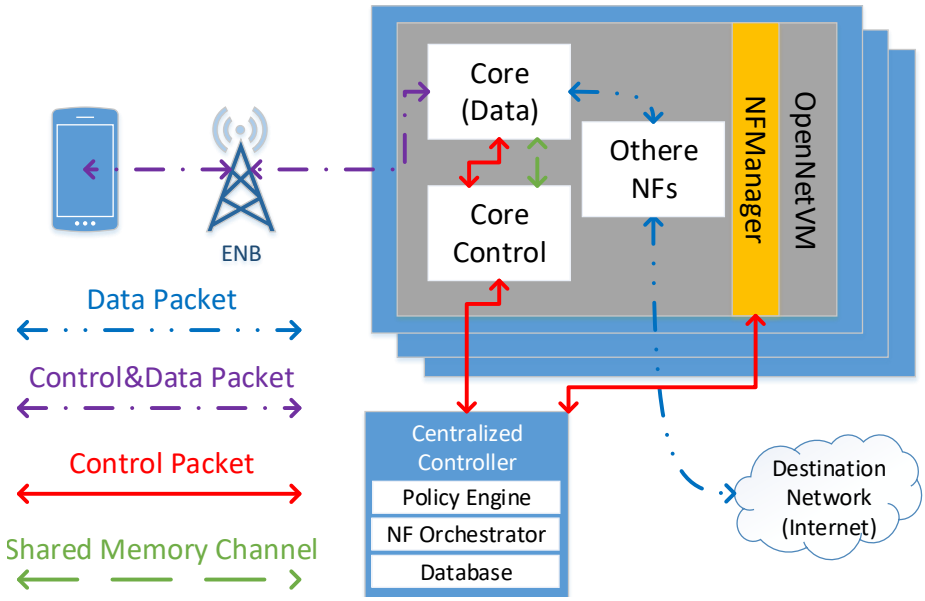


Figure 6.1: CleanG Architecture

retained in the single EPC platform (either same host or hosts within the same data center that have access to a shared database). The consolidation of the S-GW, P-GW and MME into a single software platform removes the need for a number of coordination messages between these distributed components in the current approach for implementing the EPC. The first simplification that we implement in CleanG is the elimination of GPRS Tunneling Protocol (GTP) tunnels to carry the data plane traffic from the packet backbone (e.g., the Internet) to the eNB. Because the functions of the MME and the S & P Gateways are consolidated, it is much easier to use a simple encapsulating protocol such as a GRE tunnel or a VxLAN tunnel (we use these as examples of current encapsulation mechanisms).

Additionally, other improvements to the protocol include:

- Optimizing the common case. Handling events with smaller number of messages, and

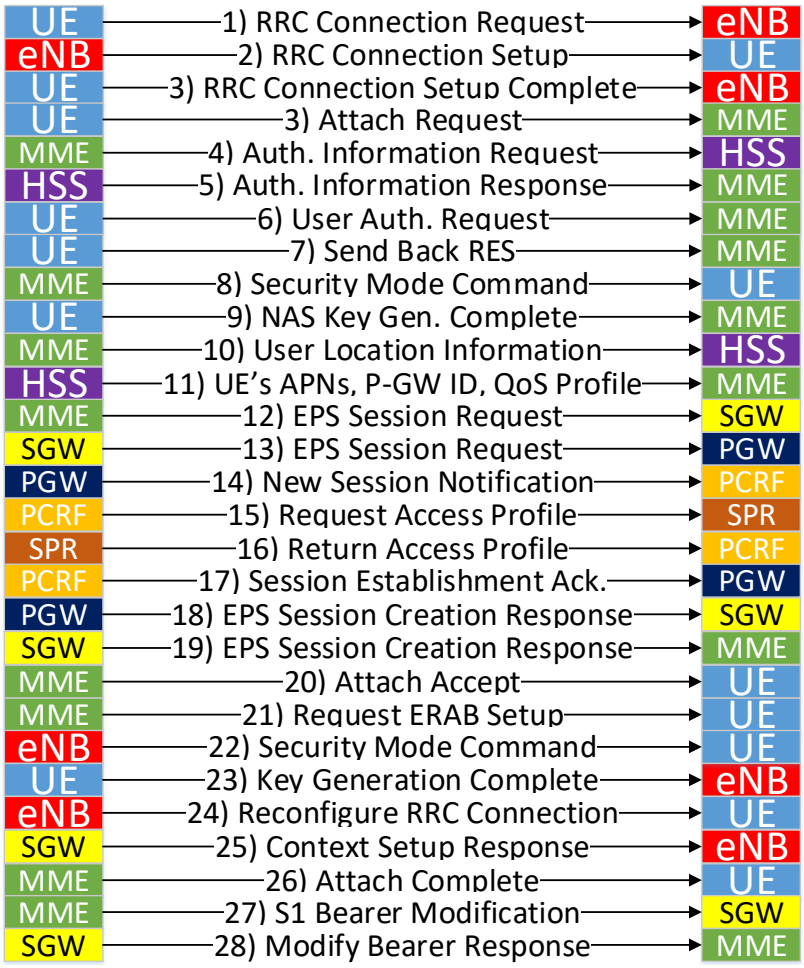


Figure 6.2: Current attach protocol

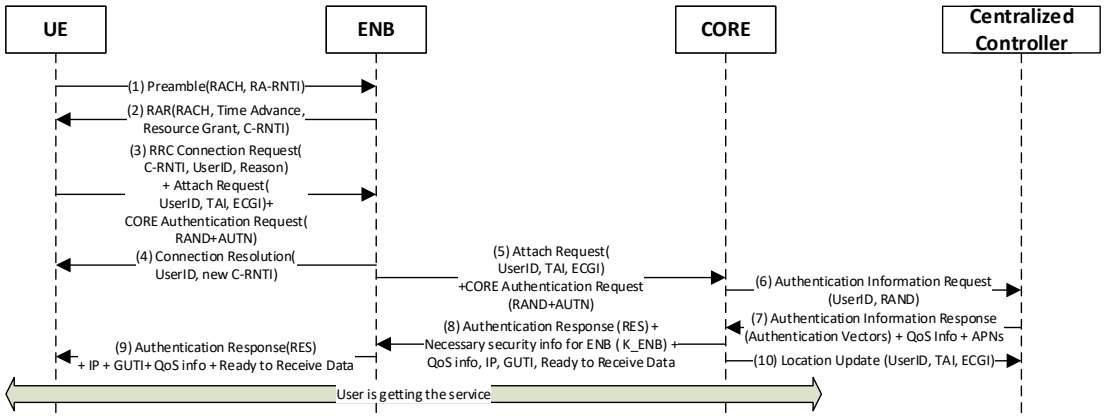


Figure 6.3: Proposed attach protocol

supporting the default, connectionless best-effort service with a minimum number of control plane transactions.

- Have a single centralized controller maintaining the data base and being high level policy decision point in the system.
- Modifying the protocol steps in certain cases. For example, having the UE initiate mutual authentication between the UE and network.
- Using a shared information base reduces message exchanges. Further, virtualization enables moving resources around (with OpenNetVM containers)

We illustrate the benefits arising from these optimizations by considering some of the most important, frequent events that are complex and messaging intensive. We first look at data packet forwarding process, which it benefits from fewer memory look ups per packet.

6.5.1 EPC Forwarding of Data Packets

Packets from UE to the packet network or vice-versa need the following functionality in the data path:

- **Having Session:** Forward a packet to a UE in the connected, active state; start the paging process if the UE is idle.
- **Detect the QoS class for a packet:** Packets need differentiated treatment based on the class of service, and if needed, be subject to charging or metering procedures.
- **Filtering packets:** A packet needs to be filtered, based on the 5-tuple in the packet

header, into different groups, with each group handled differently. (Related to the concept of TFTs in 3GPP.)

- **Forwarding packets on the right path:** Packets should be forwarded toward the UE on the right path accounting for mobility.

Packets are encapsulated by the CORE for downstream packets with a GRE header, destined to the eNB that the UE is attached to. The DSCP field in the inside header as well as the outside header are set up based on the QoS treatment required for the packet, based on a match with the packet 5-tuple. This enables the backhaul network between the Core and the eNB, as well as the eNB, once it decapsulates the packet, to provide the right QoS for the packet. Similarly, on the uplink, the UE sets the DSCP field in the packet to enable the packet to receive the right QoS treatment on the uplink.

6.5.2 CleanG Control Plane Protocol

To compare the control plane protocol for the current 3GPP architecture and CleanG, we briefly look at the protocol for attach, idle-to-active, and handover events. For an attach, the userID is used to identify and associate state with the received control plane message. Subsequently, the IP address of the UE is used to associate the control plane packets to the correct state in the Core and eNB. The eNB examines the DSCP code point of the received packet (upstream or downstream). When a session requires better than best effort service, it initially uses the default (best-effort) radio bearer, until the eNB performs admission control to elevate the session to a better-than-best-effort class. Subsequently, the eNB-UE exchange sets up the appropriate radio bearer for those packets of the UE session.

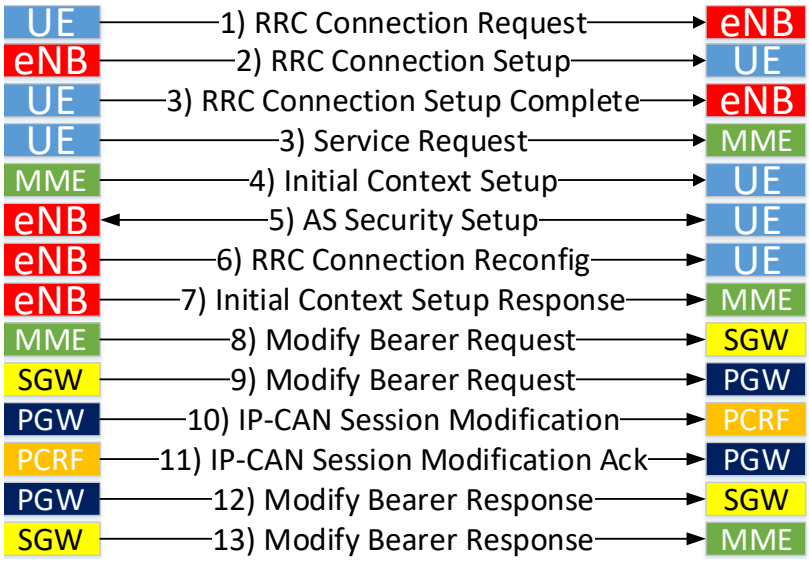


Figure 6.4: Current protocol for Service Request (Idle to Active) protocol

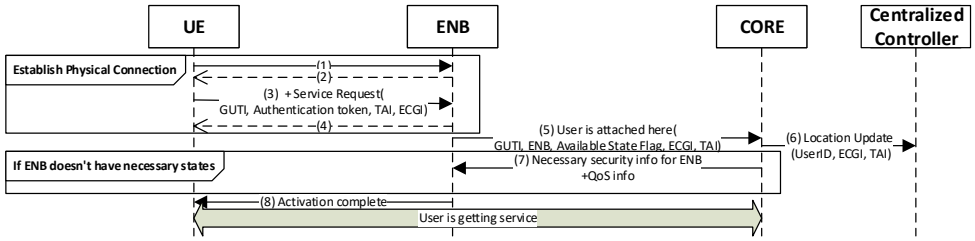


Figure 6.5: Proposed protocol for Service Request (Idle to Active) protocol

Initial Attach

Initial attach is the event when a user wants to start using the network, either for the first time or connects again after being disconnected. The high level protocol exchanges of current architecture is illustrated in Figure 6.2 and the CleanG proposed approach is illustrated in Figure 6.3. The primary distinction we observe is that the aggregation of the MME and S&P GWs results in just 2 messages between the eNB and

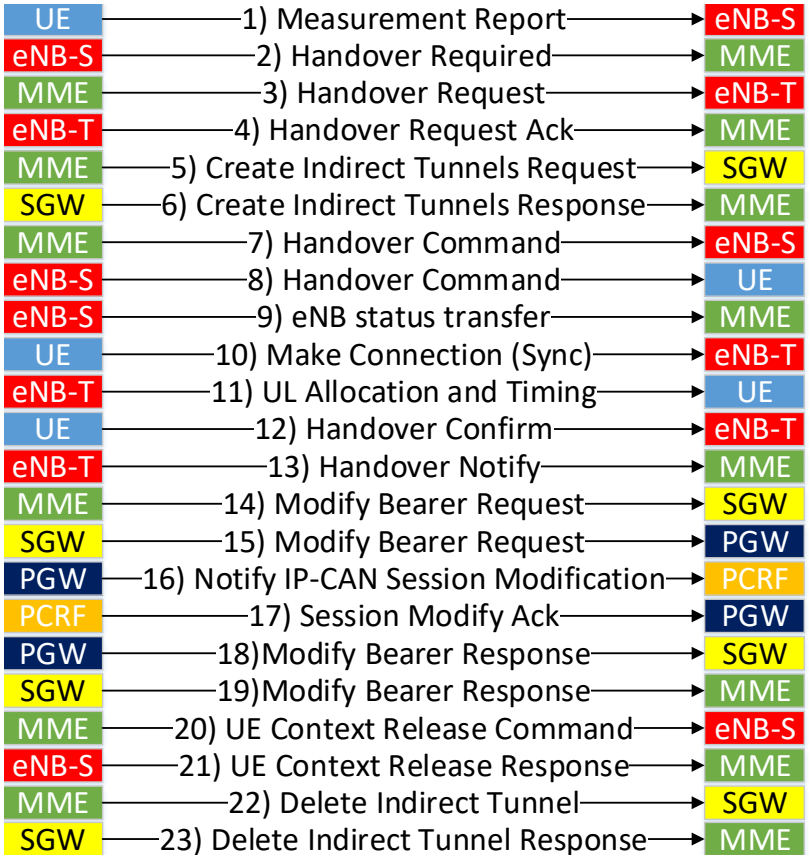


Figure 6.6: Current protocol for Handover

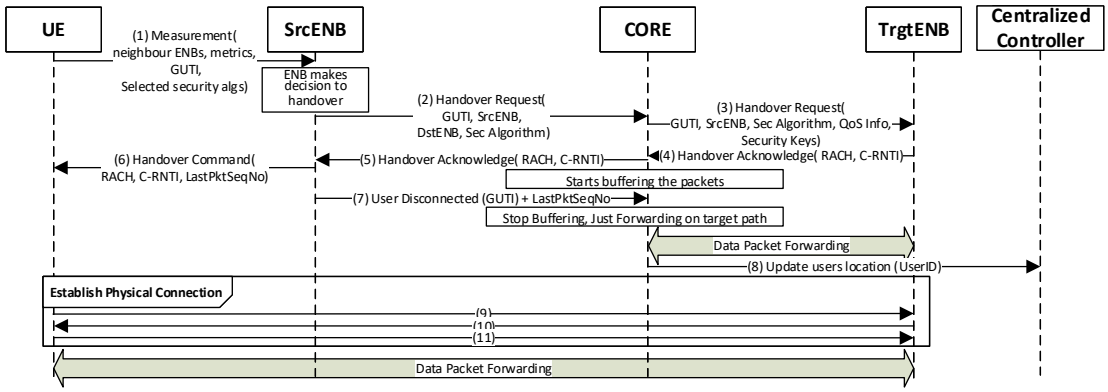


Figure 6.7: Proposed protocol for handover

the CORE and 3 between the CORE and the Central Controller/database. A number of information elements can be carried on the same control message. The primary functionality we must retain is the user authentication, which CleanG does, although the mutual authentication process is initiated by the user, and a default security scheme is chosen in CleanG. Note the significant reduction in the number of messages. Selected instance of CORE sends an authentication information request to central controller and gets, among other things, the authentication vectors(AVs) and QoS parameters. The CORE instance creates the necessary keys based on the proper AV, assigns an IP address and a temporary ID to the user (GUTI). All this information is forwarded to eNB and UE to complete the attach.

Service request (Idle-to-active)

This event triggers when UE goes from RRC-idle to RRC-connected. The crucial pieces of the idle to active procedure are the creation of a new physical connection to eNB, setting up the proper forwarding state in the Core and updating the current location of the user in centralized controller. The traditional approach is depicted in 6.4 and the CleanG approach is in 6.5.

Handover

We illustrate the complex control plane messaging required with the current LTE cellular network using 3GPP protocol for a particular (equivalent of the S1, when there is no X2 interface between the source and target eNB) handover. With the MME mediating the handover process between the S & P Gateways and the eNB, a total of at least 28 messages

are exchanged among all the entities involved in the handover (see Figure 6.6. The complex process includes messages for the MME requesting an admission control to be performed at the target eNB, state having to be set up by the MME at the source (original) and target (final) eNBs and the S-Gateway for GTP tunnels to be used for data plane communication, the handover request and response from the UE to the S-Gateway being mediated through the MME, etc., all of which become necessary when the new software EPC consolidates these functions and we optimize the common case. With the CleanG architecture and protocol, the consolidation of the functions enables the handover to be completed in just 5 messages, as shown in Figure 6.7. In detail, a UE sends its signal strength measured from received signals from neighboring eNBs to the Core. The current eNB (Source eNB) make a decision to perform the handover and send a handover request to the CORE. After receiving the handover request, the CORE starts buffering and adds a sequence number to the data packets going towards the UE. The CORE sends a request to the target eNB to which the UE needs to be handed over. If the target eNB is capable of serving the UE, it sends an acknowledgment back to the CORE instance and CORE forwards this acknowledgment back to the source eNB. The source eNB sends a handover command to UE and stops serving the UE. It sends a reroute command to the CORE. The reroute command contains the sequence number of last successfully sent packet to the UE. The core then starts forwarding packets subsequent to that sequence number to the target eNB. In the meantime, UE creates the physical connection to the target eNB and starts receiving packets from the target eNB. Thus, the entire process is not only simplified, but has a much smaller overall latency for completion. Data packets are not delayed significantly, and it results in an overall much

smoother uninterrupted data flow. For delay sensitive flows, the CORE would duplicate and forward the packets to both the source and target eNB, if backhaul bandwidth is not a bottleneck, but delay is an overwhelming concern.

6.5.3 Comparing the Overhead of Protocols

We compare the overhead of the control plane and on the data path with the existing 3GPP approach compared to the CleanG protocol for selected events:

- D1: # of control messages exchanged between the eNB and EPC components.
- D2: # of state updates impacting forwarding path in SGW (3GPP) vs. Core (CleanG).
- D3: Total # of updates impacting forwarding path across all components, EPC (3GPP) vs. Core (CleanG).
- D4: Total # of updates across all the components, EPC (3GPP) vs. Core (CleanG)

Our preliminary results are in Table 6.1, which show both a significant reduction in the number of messages as well as the amount of state updates with CleanG. Another important dimension for comparison is the number of memory lookups needed when forwarding a data packets by the EPC or CORE. While this is very dependent on implementation, and we have not yet completed a full implementation of our proposed CleanG, we have performed a preliminary estimation based on comparing CleanG to one software implementation of the existing 3GPP control plane. We believe it is possible to lower the number of memory lookups by half, which also enables faster data plane forwarding and scalability. We're working to establish this is indeed the case.

Event	3GPP				CleanG			
	D1	D2	D3	D4	D1	D2	D3	D4
Attach	19	3	4	18	5	1	1	5
Detach	7	1	2	7	2	1	1	2
Service Req.	8	1	1	8	2	1	1	2
Act. to Idle	4	1	1	4	1	1	1	1
S1 Handover	15	3	4	15	3	2	2	3

Table 6.1: Comparing Overheads of control plane protocol with 3GPP vs. CleanG

6.6 Summary and Future work

We proposed CleanG, a simplified software-based architecture and protocol for the cellular control plane for 5G networks and beyond. CleanG is based on a software-based Evolved Packet Core (EPC) which exploits virtual network functions to achieve a low-latency, responsive and scalable control plane. CleanG consolidates the EPC components of the MME, S & P Gateways, thereby reducing the control plane load significantly, without sacrificing functionality. CleanG eliminates the stateful control plane that is the cornerstone of the current 3GPP control plane. The reduced session state also decreases by half the number of lookups required while forwarding a data packet. We are implementing the CleanG architecture and protocol to do a thorough measurement based evaluation.

Acknowledgment

This work was supported in part by NSF grant CNS-1522546 and Huawei Tech. Co. Ltd.'s HIRP Grant.

Chapter 7

Re-Architecting the Packet Core and Control Plane for Future Cellular Networks

With rapid increases in the number of users and changing pattern of network usage, cellular networks continue to be challenged meeting latency and scalability requirements. A significant contributor to latency and overhead is the 3GPP cellular network's complex control-plane. We propose CleanG, a new packet core architecture and a significantly more efficient control-plane protocol, exploiting capabilities of current Network Function Virtualization (NFV) platforms. With the elastic scalability offered by NFV, the data and control sub-components of the packet core scale, adapting to workload demand. CleanG eliminates use of GTP tunnels for data packets and the associated complex protocol for coordination across multiple, distributed components for setting up and managing them. We examine the

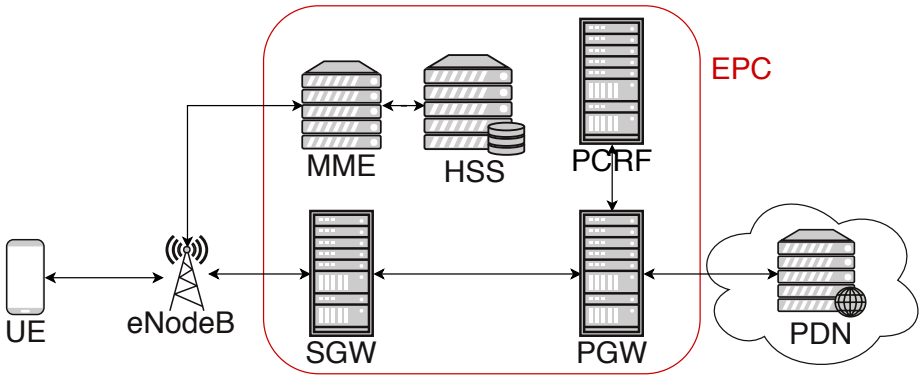


Figure 7.1: LTE system architecture

current 3GPP protocol message exchanges to carefully develop a new, substantially simplified protocol, while retaining essential functionality for security, mobility, and air-interface resource management. We have implemented CleanG on the OpenNetVM platform and perform an apples-to-apples comparison with the existing 3GPP LTE architecture, and also an architecture separating the control and user planes (like the 5G CUPS-based architecture). Testbed measurements show that CleanG substantially reduces both control and data plane latency and significantly increases system capacity.

7.1 Introduction

The architecture for the Long Term Evolution (LTE) cellular network, which has been widely deployed worldwide, called System Architecture Evolution (SAE), is shown in Fig. 7.1. The Evolved Packet Core (EPC) is normally implemented as a number of separate hardware components partly because of the need to scale control and data plane independently. This distribution of functionality among a set of distributed components

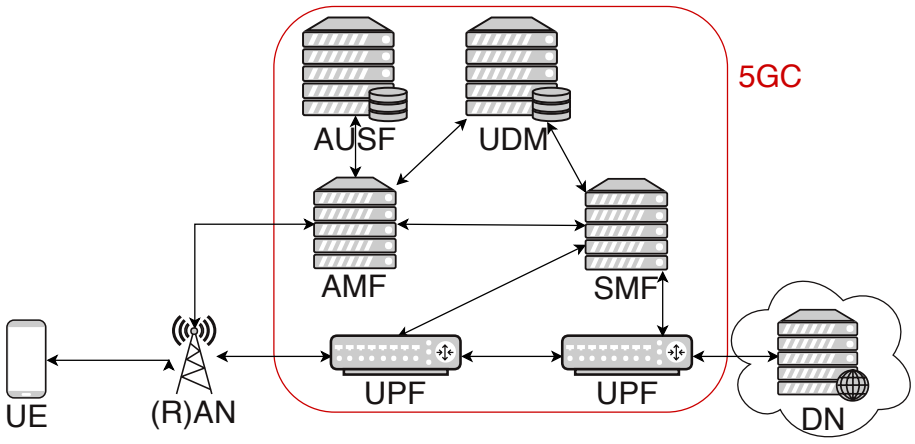


Figure 7.2: 5G system architecture

results in significant protocol overhead, especially for setting up GTP tunnels. We describe these functional components and their role in greater detail in the next section. To transition from fourth generation (LTE) to fifth generation (5G) cellular networks, in what is termed as Release 14 of the 3GPP specification, the Control and User Plane Separation of EPC nodes (CUPS) architecture was introduced. This architecture takes the separation between the control and data plane one step further. The control part of the SGW and PGW are separated from their data plane counterparts and new interfaces are defined between these new components.

The proposed 5G core architecture, shown in Fig. 7.2, is based on this CUPS architecture. The greater degree of separation between the control plane and data plane components and more granular dedication of the tasks to components may simplify each component. However, it increases the number of components involved in serving users' requests, the coordination required and the number of messages exchanged between them.

With the evolution of networks to be increasingly software-based [229], driven by

the introduction of Network Function Virtualization (NFV), cellular operators are moving towards an NFV-based packet core [93], especially for the flexibility, evolvability, and auto-scaling it offers. The progress in virtualization techniques and packet processing [240] has made it possible to host high-performance network functions on commercial-off-the-shelf (COTS) servers. Software Defined Networking (SDN) makes it easy to separate the data and the control plane, with a logically centralized SDN controller performing many of the traditional network's control functions. In this work, we explore the judicious use of software-based networking concepts to implement the core network for next-generation cellular networks. Associated with this evolution of the system architecture, we believe it is also highly desirable to re-think the protocols (especially for the control plane) of cellular networks. We believe it is critical to examine this opportunity, not just re-implement the same set of protocols in software. This paper makes the following contributions[1]:

- We rethink the role of SDN and NFV for the next generation of cellular networks. Instead of running softwarized version of the same hardware components in the 3GPP architecture as NFV components, with the SDN controller performing the mediation between the control and data planes, we design an architecture that truly leverages the capabilities of SDN and NFV. Based on the updated architecture, we propose a simple and efficient cellular core architecture, with easy scalability and support for dynamic demands.

[1]Aspect of the overall motivation and high level design of CleanG have appeared earlier in a Workshop paper [159] and as a short paper [157], which are both extended abstracts. This paper addresses the architecture and protocol of CleanG in greater detail and presents evaluation results from a testbed implementation of CleanG as well as apples-apples comparisons with alternative designs.

- In the context of such a consolidated architecture, we design a considerably simplified control-plane protocol that eliminates GTP Tunnels. Without the burden of a distributed set of cellular core entities and by leveraging shared memory across these entities, there is no need to orchestrate setting up the state for each user session. The simplified protocol eliminates a number of unnecessary control plane messages required in 3GPP, reducing both overhead (improving throughput) and latency.

- We developed an implementation of the proposed CleanG architecture and protocol (about 20K lines of code) on a testbed with the OpenNetVM DPDK-based NFV framework and examined the performance with up to millions of emulated users. The performance of CleanG is compared with a corresponding implementation of both the LTE and CUPS-based (similar to 5G) architectures. Performance measurements on the testbed implementations show the significant benefit of CleanG architecture and protocol.

7.2 Background

Fig. 7.1 shows the system architecture of LTE network. User devices/equipment (UEs) communicate over the wireless channel to base stations (eNodeBs). Their control and data packets are forwarded to the packet core components of LTE network (Evolved Packet Core or EPC). The main sub-components of the EPC are the Mobility Management Entity (MME), Serving Gateway (SGW), Packet Data Network Gateway (PGW), and Home Subscriber Server (HSS) [138]. The PGW manages IP address allocation, policy enforcement, charging, and lawful interception. Driven by software and hardware limitations of earlier

platforms, the control plane elements for handling mobility and device state management (the MME) has traditionally been separated from the data plane (S&P gateways). The MME also deals with session management and authentication of users in addition to being an end-point for the Non-Access Stratum (NAS) protocol stack between the EPC and the UE. Data is carried over virtual tunnels, called GPRS Tunneling Protocol (GTP) tunnels from the PGW to the UE. The HSS stores user information, such as security keys and the last location of the user [138]. The 5G core architecture is shown in Fig. 7.2. This architecture is conceptually very similar to the EPC with CUPS, which has the data plane part of S&P GW separated from its control counterpart. The SGW and PGW control plane duties are mostly performed by Service Management Function (SMF). SMF controls the User Plane Functions (UPFs) (which is the equivalent of the LTE SGW and PGW data plane). The AMF carries out the tasks performed by LTE MME. The Authentication server function (AUSF) replaces the MME/AAA and the Unified Data Management (UDM) is the storage component in the 5G system (in place of HSS). In the implementation of the LTE architecture, the EPC potentially comprised multiple, purpose-built hardware appliances from different vendors, motivated by the need to scale their (MME, S&PGWs) capacity independently, to adapt to the workload. In the CUPS architecture and 5G core network (5GC), the separation between the data and control plane has gone one step further, and seek to take advantage of NFV and SDN for their interaction. In the 3GPP standards-based protocol, users' packets are forwarded by using GTP-U (user plane) tunnels. Each user may have more than one tunnel set up at a time, with a GTP-C (control plane) tunnel used to create and manage these tunnels. Tunnels are identified by TEID (Tunnel Endpoint

Identifier). TEIDs are generated randomly by the receiving end of the tunnel and communicated through the MME to the head-end of the tunnel. Tunnels are used to handle user mobility and quality of service. While both these functions are necessary in the context of the 3GPP architecture, the overhead for tunnel setup and management is very high.

When we look at the current control plane protocol message exchanges for a device to attach (Fig. 7.7a) and handover (Fig. 7.9a), there are a large number of messages, with a significant number contributed by the setup and tear down of GTP tunnels. With the need for lower latency, lower overheads for the control plane, and improved overall throughput, it is essential to simplify the cellular control plane both in terms of the implementation architecture and the protocol. These are the aspects we specifically address in this paper.

7.2.1 5G Considerations

The 5G cellular networks packet core architecture has adopted concepts of software-based networking to improve scale and flexibility. We investigate potential improvements to the current architecture, the protocols for the 5G control plane and backhaul network to achieve signaling efficiencies, improve user experience, performance, scalability, and support low-latency communications. 5G networks promise to revolutionize cellular communications, with substantial increase in per-user bandwidth and low latency through improvements in the wireless radio technology. 5G net-works are being proposed as an alternative not only for traditional smart-phone based data and telephony applications but also for Internet-of-Things (IoT) and even for residential Internet service. While the use of improved radio technology will help tremendously, challenges remain be-cause of the complexity of the cellular network protocols. Of particular concern is the complexity of the control plane pro-

tocol and the use of GPRS Tunneling Protocol (GTP). Tunnels carry traffic between the end user equipment (UE) and the cellular packet core network. With the increased use of small cells (potentially more frequent handovers) and the need to support a large number of IoT devices (which switch between idle and active more frequently to save battery power), the need for efficiency of the control plane is even more important. The 5G Core (5GC) consists of several different components that carry out individual tasks. When an event for a user (e.g., attach, handover, service request) occurs, a large number of messages are exchanged between these components for notification and synchronizing state. Consider for example, an IoT device that conserves energy by quickly transitioning to an idle state, turning off the radio. A service request event (when a UE transitions from idle to active to exchange packets), requires between 13 to 32 messages (Fig. 7.4) [5]. This long sequence of messages introduces undesirable latency in initiating a data transfer after the idle period. The overhead (in messages ex-changed) and latency may nullify the purpose and goal of transitioning to an idle state. We suggest a careful re-examination of the 5G architecture and control plane protocol to improve performance. There are three aspects we explore: (a) redesigning the control plane signaling protocol and 5GC system architecture, (b) an optimized traffic engineering path selection in the backhaul network, and (c) an enhanced programmable data plane. We begin with an overview of 5GC architecture, its control plane protocol and approaches to simplify them, thus reducing latency, improving efficiency, throughput and scalability. Secondly, we propose simplification of the backhaul network, which is usually treated as an opaque entity. We explore alternatives currently being considered in the Internet Engineering Task Force (IETF). Finally, a programmable data plane is dis-cussed to

enable additional network level functions necessary for 5G applications that require high reliability or low latency.

7.3 Background

Conceptually, the 5G architecture follows the principles of the Control and User Plane Separation of cellular core (CUPS) architecture introduced in Release 14 of the 3GPP specification thus simplifying the functionality needed to be supported by each component.[178] However, the separation between the control and user plane components significantly increases the number of messages needed to coordinate a user session state across these components. There are five main entities, apart from the UE and cellular base station (called the gNB in 5G new radio) as shown in Fig. 7.3 below. These are Access and Mobility Management Function (AMF), Service Management Function (SMF), AUSF (Authentication Server Function), Unified Data Management (UDM) and User Plane Function (UPF). The UPF is a data plane entity, while the others are control and management plane entities. The AMF is main control plane orchestrator, managing UE mobility, session establishment (through SMF), and handling service requests. Additionally, entities such as the NSSF, PCF, and AF also play important roles, the details of which can be found in 3GPP TS 23.501 [213].

Traditionally, hardware components are purpose-built and customized for distinct functions. While the control plane requires capability to handle complex processing and has more sophisticated capabilities involving compute nodes, the data plane needs to perform high-speed simple forwarding and is built with hardware accelerated forwarding engines.

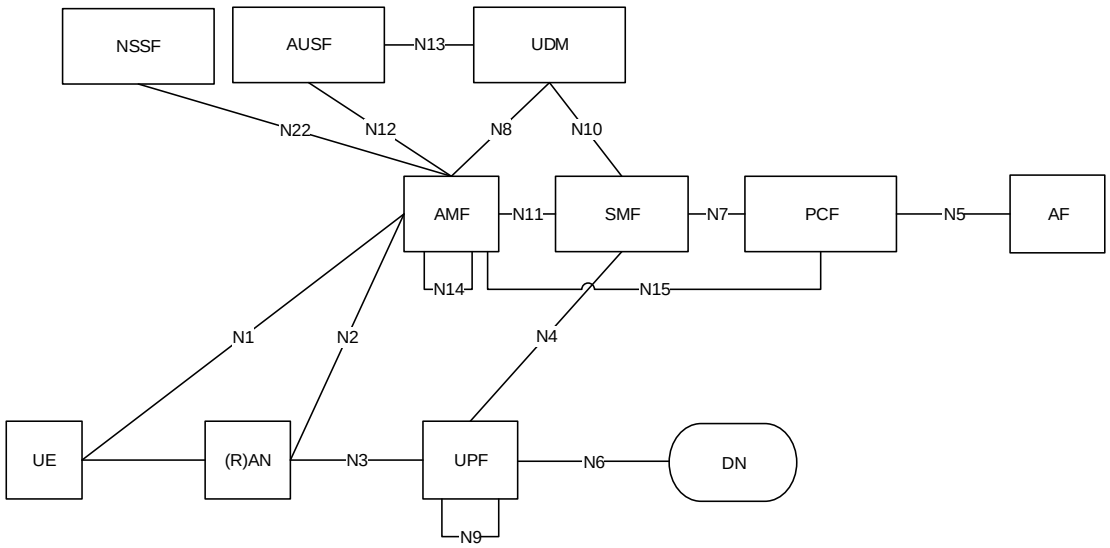


Figure 7.3: 5G system architecture [213]

However, with the advent of virtualization, common off-the-shelf server (COTS) systems with large number of processor cores, software libraries such as the Data Plane Development Kit (DPDK) and high-performance network inter-face cards, this separation of functionality is no longer necessary [55]. For example, a single server running the OpenNetVM platform can process and forward 10s of millions of packets per second with software-based network functions (NFs) handling both complex control plane functions and high rate data plane workloads [241]. This has led the cellular industry to evolve into a software-based packet core (5GC) system architecture. However, the 5G architecture continues to emphasize the separation between the control plane and data plane as one of the goals even as software-based systems are able to elegantly support multiple classes of functions running on the same system [212]. The main requirement of efficiency and high performance can in fact, be achieved by having the 5G control and data plane functions co-resident on the same COTS

system. Co-resident NFs can share state information more easily, and where possible take advantage of shared packet processing. Use of software-based NFs should be viewed as an opportunity to carefully take stock of how the functional architecture of the 5GC should be implemented. By doing so, we see the potential for re-identifying the functional components, but not require separation into different physical entities. Instead, these components can be implemented as sub-modules or NFs in a service chain (or multiple service chains) on a single system. We propose an architecture called CleanG [159], where the data plane and control plane are supported by distinct NFs collocated on the same physical system (i.e., a single component with two sub-modules). For example, supporting them on OpenNetVM platform, each sub-module can be as-signed resources (CPU and buffering) dynamically as needed based on the control and data plane workloads. CleanG remains true to 5G system architecture (release 16), both based on NFV. In CleanG 5GC functions remain logically and functionally decoupled. However, by virtue of being co-resident with other 5GC user and control functions, several messages between functions are unnecessary and communications overheads are reduced. There is no need to distribute and synchronize state information. These are major contributors to performance improvements in CleanG. Additionally, the NFV-based 5GC platform allows inherent scale-out of functions on-demand. CleanG can also conveniently support slicing of the 5G network into multiple logical slices (whether it is for having different logical planes for distinct services or for different virtual network operators) by having distinct instances of the core NF for each slice. A direction currently being pursued in industry is to have the 5G control and data planes separated by having an SDN controller as an intermediary [205]. Unlike IP networks where the timescales for control

plane updates (infrequent, of the order of seconds or more) are very different from data plane operations (frequent, of the order of microseconds or less), the cellular control plane and data plane are much more tightly coupled (e.g., when a UE transitions from idle to active, data packets can only flow after control plane operations for processing the service request are completed) [158]. Having a controller to mediate the updates between the control and user plane adds substantial delay. Additionally, the controller may become a bottleneck under heavy control traffic. Because of the need to minimize the delays between control and user plane operations in the cellular environment, we believe it is highly desirable to have them co-resident on the same node wherever possible. Supporting software-based NF offers additional opportunities for simplifying complex functions such as roaming. With the ability to copy over state of a user session, an NF can be initiated in the visiting network in a short time (less than a second). This enables the user to be served by an NF in the visiting network while maintaining information for the home network, avoiding extra packet exchanges with the home network. This approach is more efficient than normal roaming or local breakout approaches as both data and control plane components are closer to the roaming user. Our proposed CleanG architecture for the 5G cellular core exploits logically separate but physically consolidated core control (CCF) and core data plane functions (CDF), running on the OpenNetVM platform is shown below [159]. The CCF supports functionality provided by the SMF and AMF, while CDF implements functionality of the 5G networks UPF. A primary goal is minimizing delay for an update from the cellular control plane resulting in changes to the data plane. This is achieved by the CDF and CCF sharing data and state using the OpenNetVM shared memory. Finally, multiple instances

Entity	Dir	UE		RAN		AMF		SMF		UPF		UDM		PCF		UDR		N3IWF		SEAF		AUSF		EIR		DN		Total	
Message Type		B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O	B	O
Register	S	2	1	1	0	11	2	0	0	0	0	3	1	0	2	0	0	1	0	0	3	0	2	1	0	0	0	19	11
	R	1	2	1	0	10	2	1	0	0	0	4	0	0	2	0	0	1	0	0	3	0	2	1	0	0	0		
PDU Session	S	2	0	1	0	6	0	13	2	6	0	2	0	0	3	1	1	0	0	0	0	0	0	0	0	2	0	33	6
	R	1	0	1	0	9	0	9	1	7	1	2	0	0	3	1	1	0	0	0	0	0	0	0	0	3	0		
User Service request	S	1	1	3	0	3	0	4	6	2	5	0	1	0	1	0	0	0	0	0	3	0	2	0	0	0	0	13	19
	R	1	2	2	0	4	0	4	6	2	5	0	0	0	1	0	0	0	0	0	3	0	2	0	0	0	0		
Network Service Request	S	1	1	4	0	6	0	6	6	3	5	0	1	0	1	0	0	0	0	0	3	0	2	0	0	0	0	20	19
	R	3	2	3	0	5	0	6	6	3	5	0	0	0	1	0	0	0	0	0	3	0	2	0	0	0	0		
Deregister	S	1	0	0	0	3	1	3	1	1	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	10	4
	R	1	0	0	0	3	1	3	1	1	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0		
Handover	S	1	0	5	0	6	4	6	5	3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	13
	R	1	0	4	0	4	3	6	6	3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Table 7.1: Approximate number of control plane messages received (R) and sent (S) for different events in 5G. (B= baseline, O = optional messages)

of this NFV-based 5GC may be created to scale-out based on traffic, and to dynamically adapt to the ratio of control to data plane traffic.

7.4 Improving cellular control plane protocol

As the number of components in 5G increased compared to LTE, additional messages are needed to keep state synchronized among them. Table 7.1 shows the number of messages exchanged for the 5G for each user event.

Most of these messages are exchanged sequentially, or on occasion, after a timer expiration. The completion time for control plane actions for an event is the cumulative time for exchanging these messages, thus contributing to high delays. This delay includes time to process each control mes-sage and the propagation and queuing delays for sending packets between different components, especially if the control plane and data plane components are

far apart. Based on the CleanG NFV-based architecture, multiple core network processing components may be consolidated into one or more network functions running on the same node. This facilitates reducing the number of control plane messages, lowering completion time. A second major 5G overhead is in using GTP-U tunneling to carry data packets between different user plane components. The latency consuming task is the setting up of the tunnel. The receiving end assigns a tunnel ID (TEID) to a flow and notifies the sender. Because the control plane components (AMF and SMF) are involved in initiating and mediating the tunnel set up, a number of messages are exchanged, which is time-consuming. In the CleanG architecture, we use simple Generic Routing Encapsulation (GRE) tunneling that does not require explicit setup or exchanging TEIDs. Different classes of services can be used to meet simple application requirements using the DSCP (Differentiated Service Code Point) field in the outer IP header. Another challenge for future cellular networks is from new types of workload, e.g., from IoT devices. Exchanging a large number of control messages for an idle-active transition not only adds delay, but results in overhead on 5GC control components (e.g., AMF, SMF). One option pro-posed in 3GPP standardization is to piggyback data packets with the first control message to an AMF, to reduce delay. However, this can cause the AMF to become the bottleneck (excessive load from large numbers of IoT devices), contributing to additional delay. The consolidation of control and data plane components in CleanG enables immediate notification of the control plane while avoiding it having to process data packets. Consider, for example, a service request user event in the current 3GPP specification (see Fig. 7.4) for 5G networks. The AMF updates the SMF about user sessions, and receives responses. The SMF then updates the UPF, enabling

forwarding of packets by the data plane. In an NFV environment where the control and data plane components are co-resident on the same system and can share state, the need for 6a,b, 7a,b, 18a,b, 21a,b can be eliminated. Consolidation of control entities in the architecture can eliminate messages 4, 11, 15 and 19. For the common cases when an intermediate UPF is not used and dynamic policy is not enforced, 13 out of 15 core message exchanges are not essential.

5G cellular networks promise to provide low latency and high bandwidth to meet emerging, demanding, performance-sensitive applications. A key enabler is the use of NFV that offers flexibility from being software-based. We note that the use of NFV allows data plane and control plane functionality to be supported on the same platform, unlike the traditional packet core network of multiple distributed components. Scalability is enabled by the dynamic instantiation of the NFV platform supporting the CDF and CCF. However, architectural and implementation changes alone with NFV squander the opportunity for truly improving the cellular networks performance if the protocols dont properly take advantage of the ability to consolidate the tightly interdependent cellular control and data plane. We re-think the design of the control protocol to achieve low latency and high throughput by simplification and using fewer messages. Complementing this, the backhaul network and protocols must also be designed to judiciously utilize capacity and achieve low latency. The PPR protocol is a key enhancement for the cellular backhaul. Low latency applications are enabled by having a more flexible plane, using the ideas of BPP. In this article we have sought to analyze several of the complexities of cellular networks without completely disrupting the 5G system architecture, and propose carefully thought-out approaches to

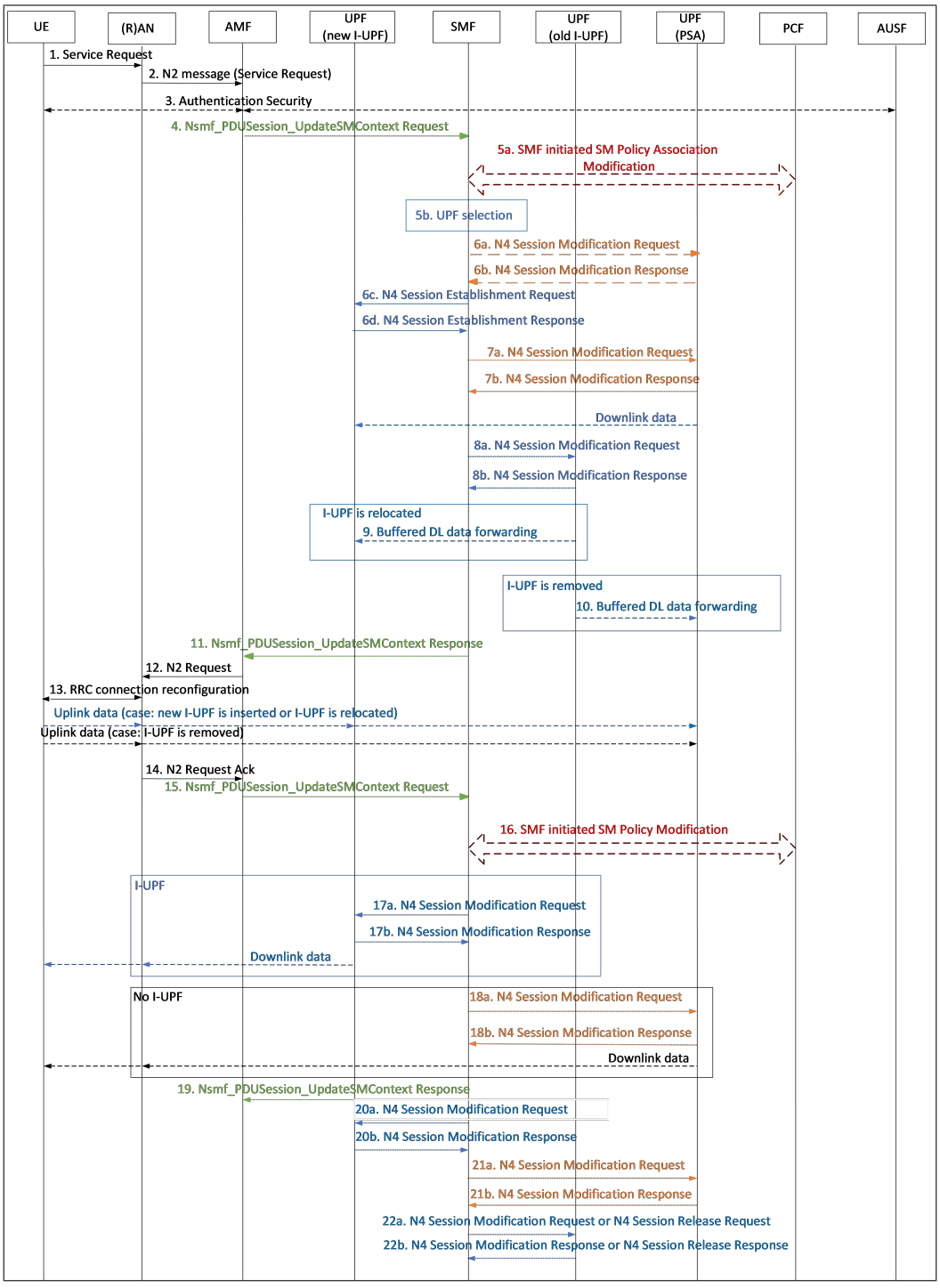


Figure 7.4: Messages exchanged for service request event in 5G among 5GC components and UE & (R)AN [5]

enhance the architecture and protocols of 5GC, the 5G backhaul and using a new backhaul transport data plane.

7.5 Proposed CleanG Architecture

The 5G architecture is meant to be NFV compatible and makes it possible to use software-based core components including utilizing the capabilities of software defined network (SDN) controllers. While we believe leveraging SDN and NFV is a step in the right direction, we believe it is possible to use them more efficiently than just simply implementing what were hardware components instead in NFV software. In LTE, the MME and S&P Gateway are separated. This separation is even more evident in 5G with the use of UPFs to handle the data plane while keeping all the control plane functions distinct and out of the data path. The ostensible advantage of this separation is that each of these components could be replicated (i.e., scaled out) to handle higher demands. While we do not ignoring some of the potential benefits of data and control plane separation, we claim this separation is not necessary or desirable in the cellular networks for a number of reasons. With current NFV platforms able to process and forward tens or even hundreds of million packets per second, they can obviate the need to solely depend on hardware performance. They can also be scaled with additional processing resources. Secondly, with the strict delay requirements for 5G and beyond, it is important to minimize the update and interaction delays between the control and data plane (as we see below, this interaction is much more tightly integrated, requiring data packets to wait till the control plane function is completed, unlike the interactions with IP, where this separation is much more feasible.) Having an SDN con-

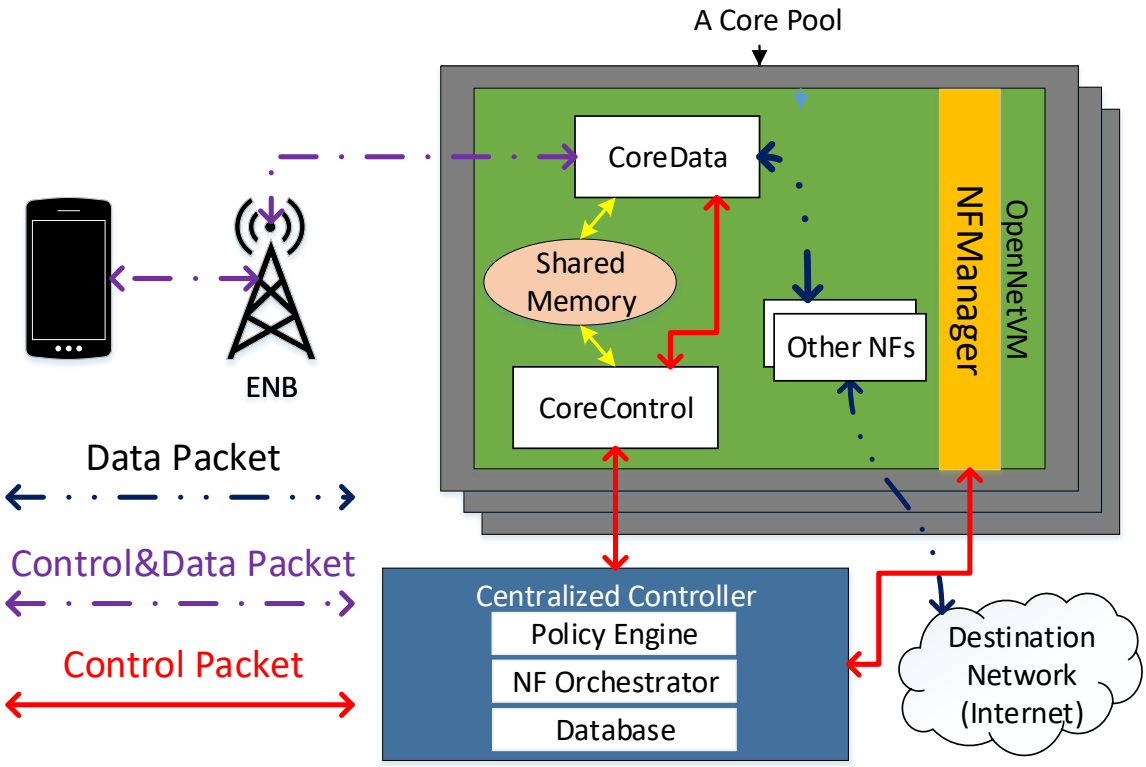


Figure 7.5: CleanG architecture

troller mediating the data plane updates (such as in [36]) only increases the delay further.

Nonetheless, we examine and evaluate the performance of the approach of using SDN-controller as in [36] to quantitatively justify our design philosophy/choice. We do believe it desirable to handle high-level monitoring and policy enforcement in a centralized manner, as with a central SDN-like controller, while keeping individual session-state 'micro-level' updates for each user transparent to the controller.

We propose CleanG as a simple, efficient, and scalable architecture for next generation cellular core networks. The CleanG architecture is shown in Fig. 7.5. The implementation architecture is based on the OpenNetVM framework [240], a high performance NFV

platform built on top of the Data Plane Development Kit (DPDK) library. A primary goal of the CleanG system architecture is to minimize the overall cellular control plane latency as well as the delay for an update to the control plane to be reflected in necessary changes to the cellular data plane. The CleanG architecture is based on having a tightly knit control and data plane for the cellular core network, running on the OpenNetVM framework. CleanG instances can be instantiated at the edge, in central office data centers, when and where demand exists. The two main components, the CoreData and CoreControl, implementing the cellular data and control planes respectively, share data and state using the OpenNetVM shared memory, buffers, and queues thus resulting in minimal delay. In our design, the CoreData data plane component receives all packets from network interfaces, encapsulates/decapsulates and forwards data packets based on the rules provided by CoreControl as a Flow table (Hash table using DPDK's cuckoo hash [111]) in shared memory. If CoreData does not have a rule for a packet (e.g., control packets), it forwards these packet to the CoreControl component. The forwarding between these components takes place without data movement of the packet, as it is achieved by adding a pointer to the packet buffer to the receive queue of the CoreControl NF. The decision to delegate the responsibility of reading all the packets from interfaces by CoreData saves resources (mainly CPU cycles) in our design. While it is normally done by the *Flow Director* NF in the OpenNetVM framework, those responsibilities are merged into the CoreData design for efficiency.

The CoreControl NF processes received control messages from the UE/eNodeB, creates a response, updates the state entry in the hash and generates a response back to the

UE/eNodeB. More detail about the control protocol and the forwarding process is provided in the protocol section.

Based on our design and facilities provided by OpenNetVM, it is possible to run multiple replicas of CoreData and CoreControl on demand, dynamically. If data forwarding is the limiting factor, an additional instance of CoreData may be created. Similarly, when handling control plane messages becomes the bottleneck (e.g., IoT environments), the number of CoreControl instances may be increased. Packets are sent to different instances of each NF, based on the Receive Side Scaling (RSS) demultiplexing provided by DPDK. Additional remaining computational capacity can host other network functions such as a firewall or IDS if physical ports are the bottleneck.

7.5.1 Deployment Considerations

To be able to handle increasing numbers of users, multiple instances of the CleanG Core can be supported on the same edge data center for the cellular provider, with a Core pool supporting a certain number of eNodeBs. The geographic area covered by each data center hosting a number of servers implementing the CoreData and CoreControl functions (Core pool as we call it) would depend on delay considerations. The more elastic the delay requirement, the easier it is to have a larger geographical area covered by a Core pool, potentially with a larger number of servers. While each eNodeB can be connected to the CleanG cores in different pools, the eNodeB would likely be configured to be associated with one Core pool based on the availability of the core instances and policies set by network providers and administrators. A user session remains associated with that instance as long as the user remains within the set of eNodeBs supported by that Core pool. We expect user

handovers between the Core pools to be infrequent, especially as there can be a relatively large number of eNodeBs connected to each CleanG Core. Thus, handovers between CleanG Core pools occur primarily to manage delay. While we provide details of the handover within a Core here, the handover between Cores is left for a detailed technical report.

7.6 Proposed CleanG Protocol

Leveraging NFV enables us to consolidate a number of components of the cellular packet core on to a single server and derive performance improvements. First, we retained the original 3GPP protocol framework (e.g., Fig. 7.7a or 7.9a) and implemented it within the CleanG system architecture. However, as we show in our evaluations, the improvement is somewhat limited, as measured by the task completion times for control events. We see a significant opportunity to dramatically improve performance when we are able to conflate the improvements from the architectural consolidation with a careful re-design of the control plane protocols to take advantage of the new architecture. Staying with the original 3GPP protocols but just changing only the system architecture (i.e., adopting NFV) results in a lost opportunity to significantly improve cellular performance.

There are two main opportunities we take advantage of to improve the cellular control plane protocol. The consolidation of the cellular core components eliminates the need to keep state synchronized among different components and the consequent need for a number of additional messages to be exchanged to confirm the state update. A second major improvement comes from eliminating the process of setting up the GPRS Tunneling Protocol (GTP) tunnels by using the simpler Generic Routing Encapsulation (GRE) tunnels. To

eliminate GTP tunneling, while supporting different classes of service (CoS), we use the DSCP header field in the encapsulating IP packet to specify the required CoS (bearer in 3GPP cellular network terminology). Details are in subsection 7.6.1

In addition to the techniques we already mentioned, we use the following measures to improve the control plane protocol: (details for individual events are explained in the subsections below.)

- We optimize the protocol for the typical scenario and then address exceptions instead of burdening the common case with unnecessary messages. For example, by knowing the security algorithm previously used by the user, we take advantage of this soft state for the user when he uses the same algorithm for the next connection. If however the user changes the algorithm for the new connection, an extra message is exchanged to change the selected algorithm.

- The central controller is only used for high-level monitoring and policy enforcement. The controller is not involved in the exchange of each and every control message.

- Where appropriate, we take advantage of changing the order of the messages exchanged to reduce the number of messages. For example, mutual authentication needs a minimum of three messages, but by initiating from the client, we can reduce the number of messages exchanged.

- We can merge information across what would otherwise have been carried across multiple messages where appropriate.

- Where appropriate, we delegate responsibility to other network components. For

instance, we allow the eNodeB to participate in the authentication thus reducing the need to send an additional message back to the CoreControl.

- By taking advantage of shared memory between the cellular core components, we reduce the need for the exchange of several messages, and the shared data structure allows for synchronization and sharing of information.

- Based on the user service required (e.g., delay tolerant or delay sensitive), events are handled differently. For example, for handover of a delay sensitive stream, packets are duplicated. But, they are not duplicated for the delay tolerant streams.

- If not necessary, control message exchanges are deserialized. For example, location can be updated in the HSS while the attach acknowledgment is sent to the UE in parallel.

7.6.1 Forwarding data packets in CleanG

Forwarding in the 3GPP protocol used in LTE & 5G is described in detail in [3, 4]. For each CoS of each user, a distinct tunnel is established between the entities (SGW, PGW, and eNB in 4G, and between UPF and gNB in 5G) to forward data packets. Tunnel IDs are exchanged between these entities, mediated by the control plane components. For example, to forward packets from eNB to the SGW, the SGW sends a Tunnel End Point Identifier (TEID) to MME, and the MME forwards that TEID to the eNB. All packets for that stream are encapsulated in a packet with that TEID. When the SGW receives packets with that TEID, it fetches the QoS, and handles the packet accordingly. However, in our CleanG

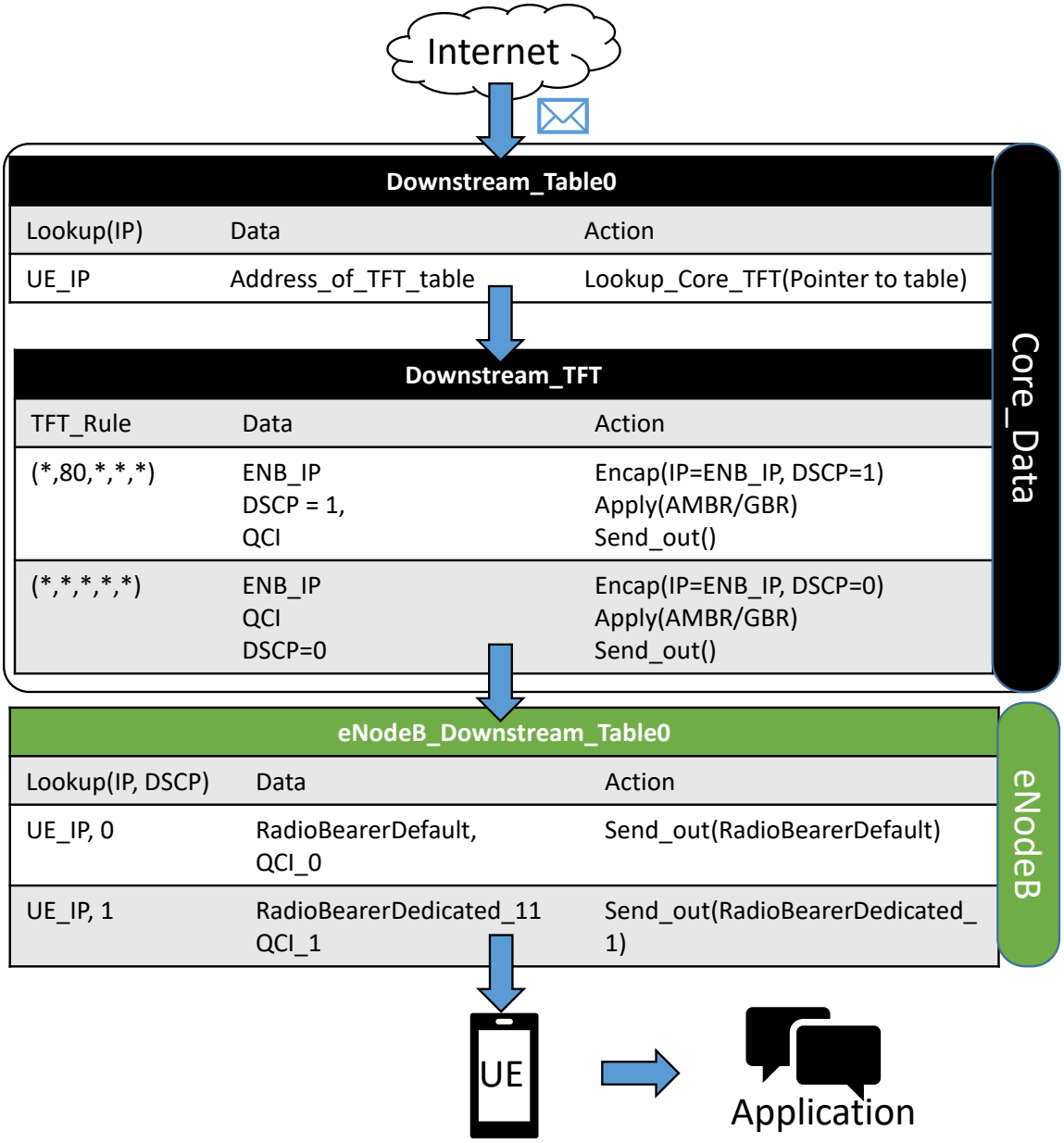


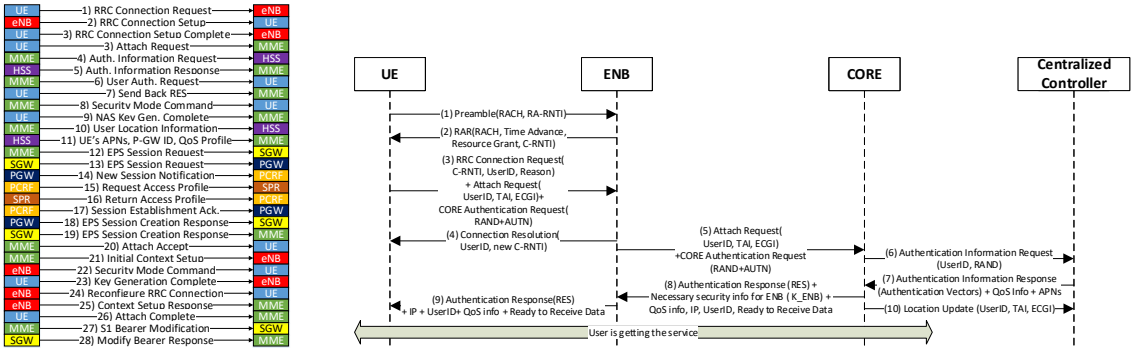
Figure 7.6: Downstream forwarding tables

protocol, we dispense with the use of TEID, because the CoreData NFV can forward data packets based on the IP address and QoS (DSCP) fields in the IP packet. Another benefit of GTP tunnels was for handling mobility. However, mobility can be dealt in CleanG by attaching the right GRE encapsulating headers to data packets. GRE tunneling has much less overhead for the system as it does not need a TEID setup step. The CoreData and eNB simply use the IP addresses of each other to forward packets to the other end. We now explain this in detail. While we retain QoS treatment and admission control for the data plane, there are a number of significant differences in CleanG.

Fig. 7.6 presents abstractly the forwarding tables for downstream (From data network to UE) packet processing. Based on the destination IP address, the UE's IP, a Traffic Flow Template (TFT) of the user is fetched. The packet header is matched against the TFT rules and based on that, the class of the service for flow is selected. Besides that, we obtain the IP of the destination eNodeB, which is then is used to encapsulate the packet with a Generic Routing Encapsulation (GRE) header. Based on their class, packets will be forwarded with the appropriate priority by CoreData, with the Differentiated Services Code Point (DSCP) field set accordingly. DSCP is used by routers in the backhaul network to forward packets and the serving eNodeB to select the proper radio bearer.

The third table in Fig. 7.6 is the forwarding table in the eNodeB. Based on the IP address of the user and the DSCP bits, the proper radio bearer is chosen and the packet forwarded to the user.

For the upstream, the UE receives the TFT rules during the initial attach process, which is used to match packets from the application/higher layer protocol stack on the



(a) Current LTE protocol

(b) Proposed CleanG protocol

Figure 7.7: Attach protocol

UE. If a packet matches a rule, but the bearer has not been already set up for it, the UE sends a request to setup this bearer. This request to set up the bearer can be handled by the eNodeB or be sent to the CleanG core based on provider policies. The UE then uses the radio bearer to send that packet based on the TFT table. The eNodeB receives the packet, based on the IP and radio bearer, sets the value of the DSCP bit and encapsulates the packet to be sent to the CleanG Core. The CoreData NF decapsulates the packet, serves it based on the service class, and forwards it to the destination. To sum up, in CleanG, instead of creating separate GTP tunnel headers (tags) for each user and class of service, and exchanging a number of messages between the different components to set up the tunnels across different components, we use an encapsulating IP header and DSCP bits to provide that functionality, without requiring set up across multiple components. This is far more efficient, without needing *any message exchanges.*

7.6.2 CleanG control plane protocol

While we *ensure the user/UE functionality is retained*, we deviate from strict conformance with the 3GPP protocol (e.g., NAS, S1-AP, etc.) to achieve what we believe are compelling performance benefits that come from re-thinking the control plane protocol in addition to the architectural changes of CleanG. Messages are exchanged over a reliable transport protocol (SCTP or TCP). We explain the protocol for each of the following main events, looking at their primary variant: initial attach, service request, handover, active-to-idle, and detach.

Initial Attach

This is the event when a user wants to start using the network. In contrast to the current 3GPP protocol for Attach (Fig. 7.7a), the CleanG message exchange is shown in Fig. 7.7b. We retain the physical connection setup between UE and eNodeB as in 3GPP [138], with its four messages for the initial attach request and establishment of the physical, air-interface connection between UE and eNodeB. We initiate the authentication procedure by the UE rather than the network, thus allowing the authentication request to merge with the attach request and reducing the number of messages exchanged. (Omitting authentication request and response, originally message numbers 6 and 7 in Fig. 7.7a). The reason this change is valid is that the authentication process between user and network is mutual authentication, and either end can start the procedure. A sequence number is used to track the freshness of message exchanges. It is also used to create the authentication vectors at the logically centralized controller. Another option, instead of the authentication

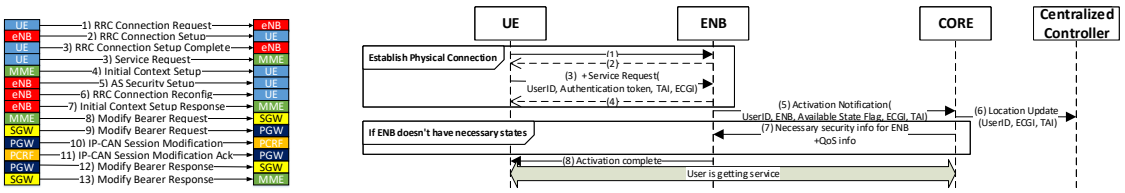
process being started by UE, is to delegate part of the responsibility of authentication to the eNodeB. Here, the expected authentication response is provided to the eNodeB, which then validates the authentication response from the UE. The default assumption is that these values should match. However, if the authentication was not successful, the eNodeB would send a message back to the Core and informs it to ignore the authentication request, otherwise the Core assumes the authentication was successful.

Another improvement from a security perspective in the CleanG protocol is that for each user or group of users, a default encryption and message integrity algorithm is specified. However, if it is not possible to use the default algorithm by network or UE, it is possible to use a different algorithm on demand by signaling the other side. As such, the message exchanges between components to negotiate the security protocols can be omitted so that messages 8, 9, 22, and 23 in the 3GPP protocol (Fig. 7.7a) can be eliminated.

When the HSS sends authentication information to the Core it also includes other user-related information such as the data network, QoS for the user. Therefore, it is not necessary to have a dedicated message from HSS to the CleanG Core to carry this information.

Because of the consolidation in CleanG resulting in a session's data and control plane functions being on the same machine, message exchanges between MME, SGW, and PGW of the 3GPP protocol are not necessary, eliminating another group of messages 12, 13, 17, 18, 19, 27, and 28.

A central controller can enforce the Policy and Charging Rules Function (PCRF) and Subscriber Profile Repository (SPR) related updates and policies, thus eliminating the



(a) Current LTE protocol

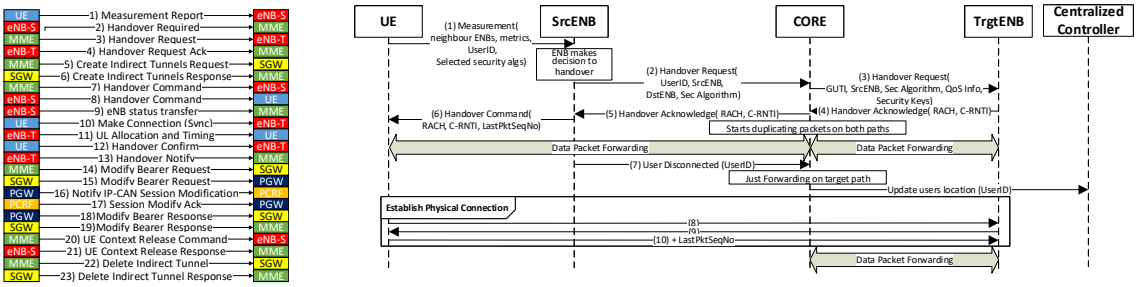
(b) Proposed CleanG protocol

Figure 7.8: Idle-to-active protocol

messaging for these function. The central SDN-like controller may monitor the system and update the policies on demand. For example, in response to network congestion, it can update the policy in the related Core pool to re-prioritize traffic appropriately, or modify shaper parameters. These updates are not sent for individual users. Instead, they are carried as policies that can be applied on classes of users.

Service Request (idle-to-active)

The idle-to-active transition occurs when a user that was not active for a while, begins to exchange data (Fig. 7.8). The first four messages are exchanged to establish the physical connection and sending the idle-to-active request, which are unchanged in CleanG. Then, in CleanG, the eNodeB sends a flag about its current state in addition to the notification to the CoreControl component. If the eNodeB already has the required state to handle user packets based on the last time the user was active, CoreControl does not need to send the state back to the eNodeB, but just sends back an acknowledgment. For selected users (chosen by CoreControl, e.g., an IoT device), if the eNodeB has the state, instead of explicit messaging to notify the idle-to-active transition, the users can send their



(a) Current LTE protocol

(b) Proposed CleanG protocol

Figure 7.9: Handover protocol

packets directly. Then, CoreData may just forward the data packet or notify CoreControl, so the data packet is considered as an idle-to-active request as well. If the location of the user hasn't changed, it is also not necessary to update the user's location.

Handover

One of the more important protocol events in the cellular network is a handover from one (source) eNodeB to another (target) eNodeB. Since it is frequent and a large number of messages are exchanged, handovers can impact user-perceived latency and significantly impact the performance of data packets. In the current 3GPP protocol, handover is agnostic of the type of user data from the cellular core's viewpoint (it is handled differently in the Radio Link Control (RLC) layer by using 'acknowledged' or 'unacknowledged' mode, but we do not focus on this aspect in this paper). CleanG recognizes a need to differentiate between delay sensitive packet and delay tolerant packet classes to optimize the handover appropriately. Fig. 7.9b shows our proposed CleanG protocol for handover of users having a delay sensitive packet stream. The initial handover preparation is similar

to what is used in current S1 3GPP handover procedure. However, the most important difference is that instead of setting up indirect tunnels, CoreControl begins duplicating the downstream packet towards both the source and target eNodeBs, while adding a sequence number to the packets. This allows significant latency reduction, at the cost of overhead in duplicating packets over the backhaul network (which is most often not the bottleneck and is provisioned for peak capacity, if not more). It allows, at the point of handover execution, when the UE is connected to the target eNodeB, all the packets are buffered there and are ready to be delivered to UE. The UE provides the last sequence number of packets received from the source eNodeB as well as other flow state information needed at the target eNodeB to provide correct delivery of the packet stream to the UE. The target eNodeB from that point on delivers packets in sequence to the UE. The packet duplication occurs only during a short during of the handover from source eNodeB to the target eNodeB (after the determination has been made to initiate the handover) for users with delay sensitive streams over the wired backhaul. The benefit is the significant latency reduction. For users with delay tolerant streams, instead of the duplication to the target eNodeB, CoreControl buffers packets at the Core. Once the user is attached to the target eNodeB, the packets are transmitted from the CoreControl to the eNodeB. The remaining protocol exchange for both delay tolerant and delay sensitive sessions are the same. Overall, the procedure is dramatically simpler than the current 3GPP procedures.

Active-to-idle and Detach

Active-to-idle and detach are used for short term and long term service suspension respectively. While we have made similar improvement to their protocol, the details are omitted for the sake of brevity in this paper.

7.7 Evaluation

We compare our proposed CleanG with the 3GPP LTE architecture and with a CUPS-based (similar to 5G) architecture and protocol. With the CUPS-based architecture, an SDN controller mediates communication between the data and control plane entities (we interchangeably call this the SDN-based alternative). Without an open source implementation for 5G and with limitations of existing open source implementations of the EPC for LTE, e.g., OpenAirInterface [167], we chose to implement each of the three variants as carefully and fairly as possible on the high performance OpenNetVM DPDK-based framework. In this way, we perform an apples-to-apples, fair, comparison of our proposed CleanG system, with other approaches. *All* three architectures and protocols were evaluated in the OpenNetVM framework, ensuring all of them benefit similarly from OpenNetVM's features, including DPDK-based packet processing. As a consequence we are able to clearly show the performance improvement of CleanG over the alternatives is based on the architecture and protocol improvements and not because of the use of a faster platform. In our implementation, we included all the key messages and the primary information fields that are key to the protocol operation. The protocol and system implementation used C (about 20K lines of code) to obtain the highest performance we could obtain for all three different

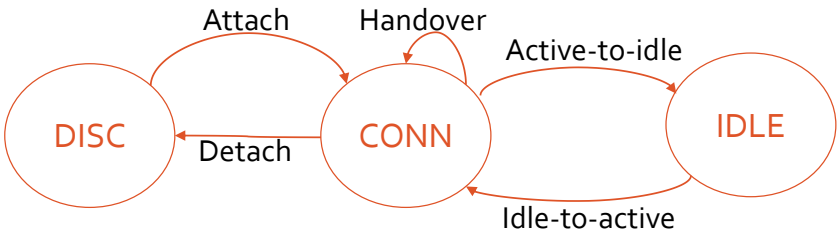
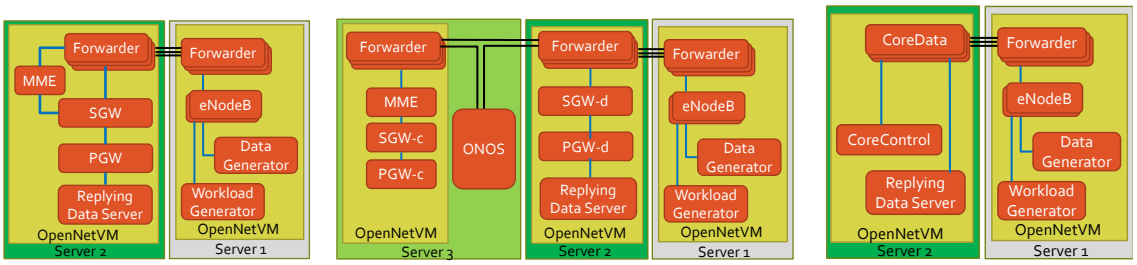


Figure 7.10: Transitions for each UE at workload generator

architectures. A small part of the code for the CUPS-based approach that runs over the ONOS [28] SDN controller is in Java. We use one server to generate the user workload traffic at scale. (Server 1 in Fig. 7.11). Based on reasonably representative user behavior, we are able to scale the system up to support millions of users. The Forwarder NF in Fig. 7.11 is a foundational, base NF that is needed to read packets from ports and forward them to the appropriate destination NFs. We avoid simulating the air interface in detail by modeling the traffic generated from a large number of UEs to the eNBs. We generate a number of representative user events (e.g., UE connecting/disconnecting to/from the network, the UE going to idle state, mobility causing handover, etc.) for each UE emulated. These events are generated and arrive at the NF representing the eNodeB. The workload generator maintains state for each UE to generate the appropriate messages and the UE changes state based on the state machine shown in Fig. 7.10. This state machine is used to regulate the transition rate between states for each user and the number of events generated by each user. This state machine is valid even considering the Discontinuous Reception (DRX) state transitions [2], as it can be interpreted as a sub-state in the connected state.



(a) LTE EPC Implementation (b) CUPS-based Implementation of the EPC (c) CleanG Implementation

Figure 7.11: Components involved in implementation of each scenario

Users can be in one of three states: DISConnected, CONNected, or IDLE. Five different transition rates are defined. We set these rates based on [204], assuming users spend 10% of the time in CONN and DISC states and 80% of the time in the IDLE state. Another parameter driving the workload generator is the number of control transactions per second per user. We use as the basis the number of transactions measured in 2012, which was 0.031 per second per user [1]. We extrapolate from that data to arrive at an anticipated current workload (we recognize there will likely be inaccuracies as usage and deployment pattern change along with new applications). With a 2x increase every three years, we arrive at a rate of 0.1 transaction per second per user. By considering the amount of time spent in each state and the total number of transactions for each user, we calculate that the transition rates are 0.07/s for attach, handover and detach, 0.215/s for active-idle and 0.0269/s for idle-active transitions. Another parameter needed for the evaluation is the number of data packets sent by each active user. We set it based on the assumption of total data usage for each user (about 2GB per user, given the average amount of traffic per smartphone in 2016 was 1,614 MB per month [50]) and the percentage of time each user is

active was 10%. From this, we get 108 Kbps of data traffic on average for an active user. Using 617 Bytes as the average packet size [110], each user exchanges about 21 packets per second. [204] has measured that smartphone users have a higher downstream rate and upstream (as can be expected). Hence we set a 2:1 ratio between downstream and upstream traffic. Thus, we use 14 pkts/sec downstream, and 7 pkts/sec upstream for each active user.

The LTE core implementation on our testbed is illustrated in Fig. 7.11a and the implementation of SDN-based and CleanG are shown in Fig. 7.11b and Fig. 7.11c, respectively. For emulating the LTE network, each component of the LTE Core is implemented as a distinct NF in OpenNetVM, in addition to the Forwarder NF. Although the MME is usually not co-located with SGW & PGW on the same physical node, we have chosen to have it on the same node to have the most efficient implementation of LTE EPC core, with minimum delay, and thereby ensure a fair comparison with CleanG. We note that the more conventional LTE EPC core partitioning would result in more delay and lower performance for the competing alternatives than we would obtain here. In this way, we can fairly evaluate the effects of architecture and protocol of LTE vs. CleanG.

7.7.1 Total number of supported users

We first measure the maximum number of the users that can be handled by each architectural alternative for the same workload of data and control traffic, providing the same resources. We increase the number of users by 50K in steps until packet drops are observed. Other parameters such as queue size and buffer size are the same across all the experiments and systems. We start by having a CPU core for each function for each option - that means, we use 4 CPU cores for LTE (one for MME, SGW, PGW, and forwarder),

while with CleanG we use a CPU core each for CoreData and CoreControl. The maximum number of users supported by CleanG with the one CPU core hosting CleanG CoreData is 1.65M (Fig. 7.12). In comparison, the LTE architecture and protocol supports 1.45M while the CUPS-based alternative can only support 1.15M users.

We also look at the alternative of using an equivalent amount of resources for CleanG as the other architectures. When CleanG uses the same number of the CPU cores as LTE (i.e., 3), we see that it can support a significantly higher number of users, going up to 5M users, resulting in a 3-fold increase in the number of users compared to having one core for CleanG CoreData. This is feasible since the NF implementing the CleanG control plane (CoreControl) on one core is able to support the corresponding load of control messages. The 3 extra CPU cores can thus be assigned to the CoreData instances. We also note that CoreData is the bottleneck. In the LTE case, the SGW becomes the bottleneck, as it has to handle the highest number of messages, including both data and control messages. In the CUPS-based implementation, however, the SDN controller is the bottleneck since it is limited in handling the large number of control messages received.

7.7.2 Maximum data plane rate

The maximum data plane packet rate that can be handled by each approach is shown in Fig. 7.12. The CUPS-based approach can support a higher data packet rate than the LTE architecture, because of the separation of the S&P GW data plane functions from the control plane functions. The control plane functions are handled by the components residing on the north-bound side of the SDN controller. This improvement comes at the

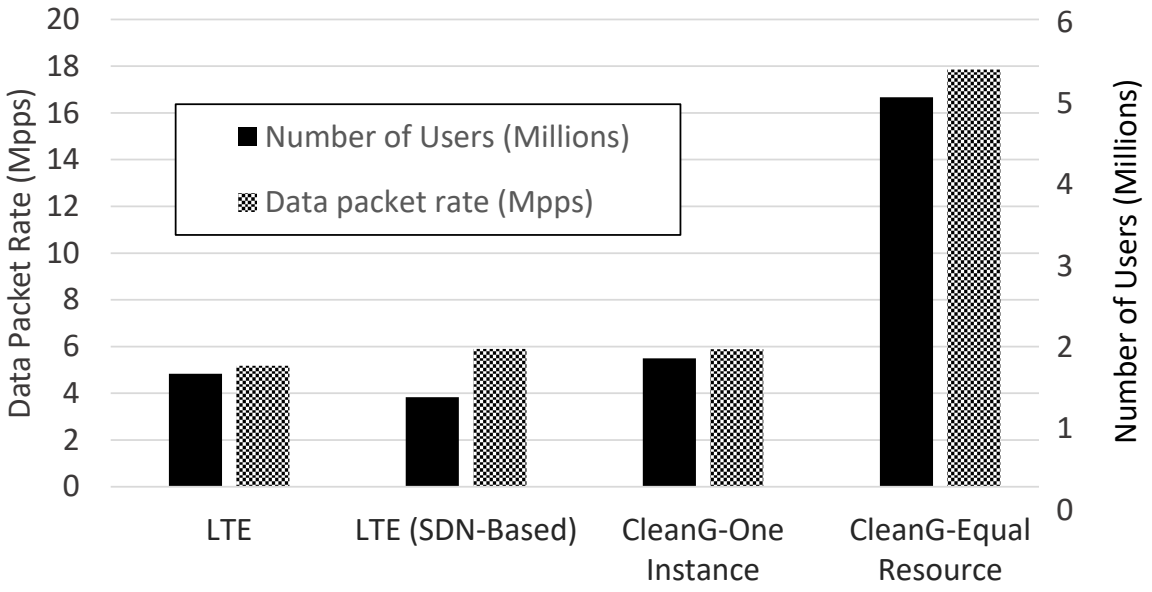


Figure 7.12: Maximum data packet rate in Mpps and maximum number of supported users

cost of reducing capacity in terms of the number of users supported as a result of control plane limitations. On the other hand, when we compare these with CleanG when provided with an equal number of processing cores, CleanG is at least 3X better in terms of the supported maximum packet data rate, demonstrating the benefit of both architecture and protocol simplification.

7.7.3 Total UE event completion times

An important measure of the efficacy of the control plane is the time it takes to complete actions requested by a UE. Typically, data packets exchanged between the UE and the rest of the network have to wait till the control plane protocol sequence (often involving several packets) is completed. Attach and Idle-Active transitions cause data packets to be buffered at either end. "Active-Idle" and "Detach" event completions result in a faster

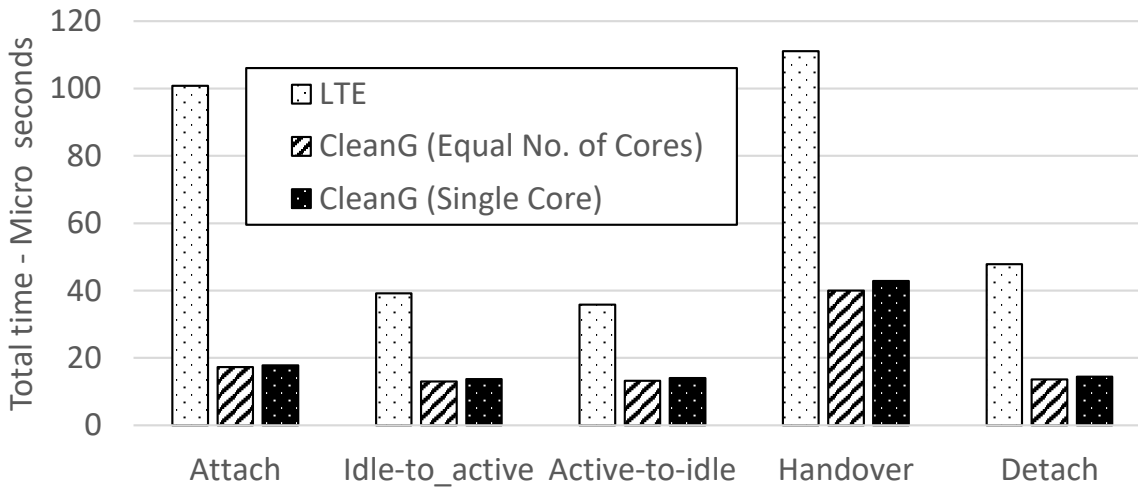


Figure 7.13: Events completion time for 1M users

release of resources (and in the former, better battery power management). Faster handover causes less disruption in the data stream.

Fig. 7.13 and Fig. 7.14 show the completion time of the different events for a system with 1M and 100K users respectively. CleanG has a dramatically lower delay in comparison to the other two approaches, even when it is using fewer resources (i.e., having only a single CPU core). By using an equal amount of resource as LTE, the completion time for the control plane action is slightly faster with CleanG. Fig. 7.15 shows the completion time for the CUPS-based approach (shown separately because of the larger Y-axis scale), which is noticeably higher than the other two approaches because of the delay imposed by the control messages having to traverse the SDN controller. Since the data plane is the bottleneck with CleanG, the addition of CPU cores to the control plane primarily improves system capacity (number of user sessions) rather than improve event completion times. This

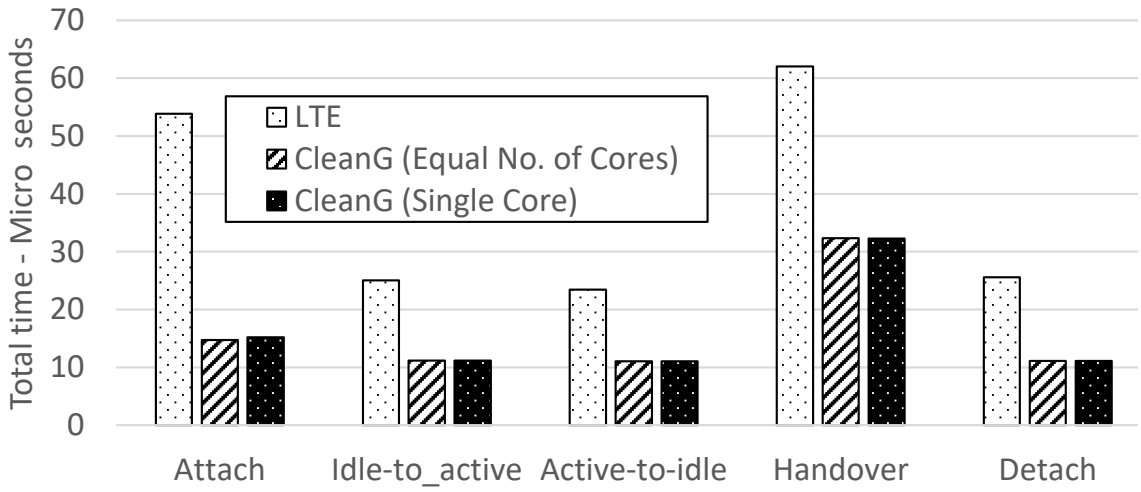


Figure 7.14: Events completion time for 100K users

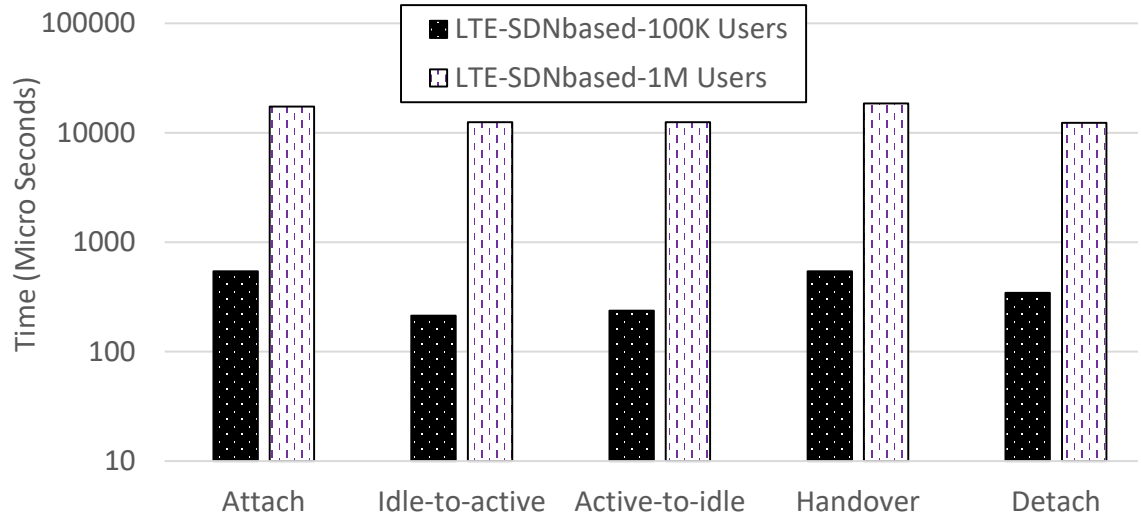


Figure 7.15: Events completion time for SDN-based

observation is based on the workload's mix of data and control traffic we used. It would be different if the control traffic had a higher intensity.

7.7.4 Data packet forwarding latency

Forwarding data packets by the cellular core is affected by the load on the different components as well as the control plane protocol. A more efficient architecture and protocol not only improves the performance of the control plane but also reduces the latency for data packets. Fig. 7.16 shows the cumulative distribution function (CDF) of the round trip time (RTT) from the data generator to the data server and back, as observed by data packets for the different architectures. This graph shows that data packets in CleanG see lower delay by at least a factor of 2 in comparison with data packets in the LTE architecture. By increasing the number of cores for CleanG, to match available resource for LTE, the difference becomes even more significant. Finally, we observe that the data forwarding latency for the SDN-based alternative is comparable to the CleanG approach. This is because the data forwarding components are not involved in handling control messages in the SDN-based alternative.

7.7.5 Detailed timing of different events

Each of the main events has a number of messages exchanged between the different components. We recorded timestamps on server 2 and 3 (see Fig. 7.11) to calculate the time spent on processing and transmitting each of the control plane messages. Although we captured the timestamps comprehensively, we highlight the major components of time spent

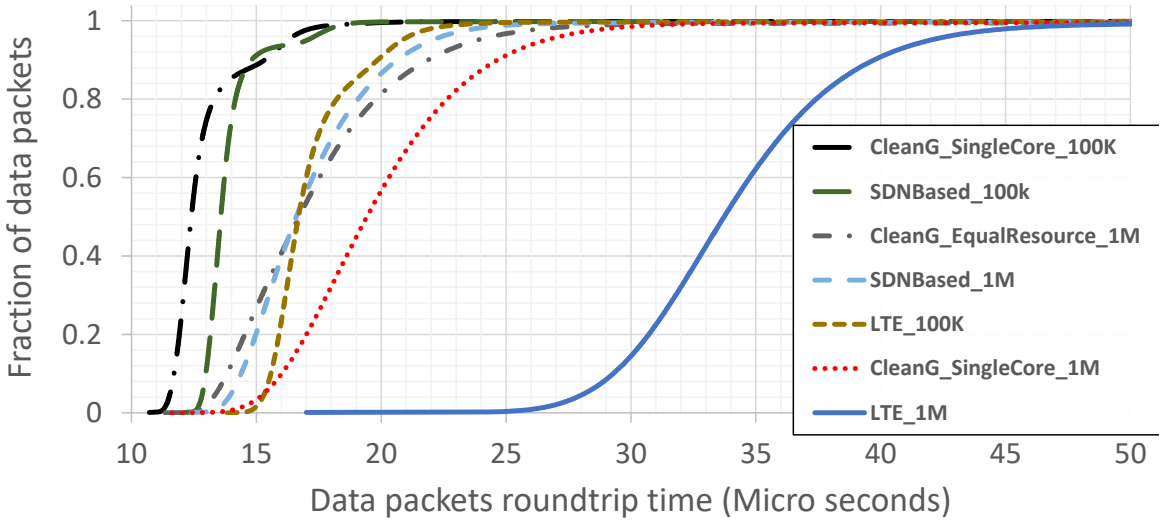


Figure 7.16: Data forwarding delay comparison

on each event to help understand the contributing factors. As seen in Fig. 7.17 for LTE, the Attach and Handover events exchange a much larger number of messages compared to the other 3 events, thus taking more time, as seen in Fig. 7.13 and Fig. 7.14. Each column shows the time spent for each of the control events. Three different categories of messages taking distinctly different times are observed. The more significant chunks of time in Fig. 7.17 represent the time spent on messages exchanged between components residing on different physical devices, because of the additional time to exchange messages between the servers over the link. We underestimate these completion times, because if we had the components such as S&P GW and MME on different machines, the overall delay would be even higher.

But what is worse is the even higher delay observed for the CUPS/SDN-based architecture, as shown in Fig. 7.18. The largest time spent is communication between the data and control plane of S&PGW when the messages are forwarded using an SDN

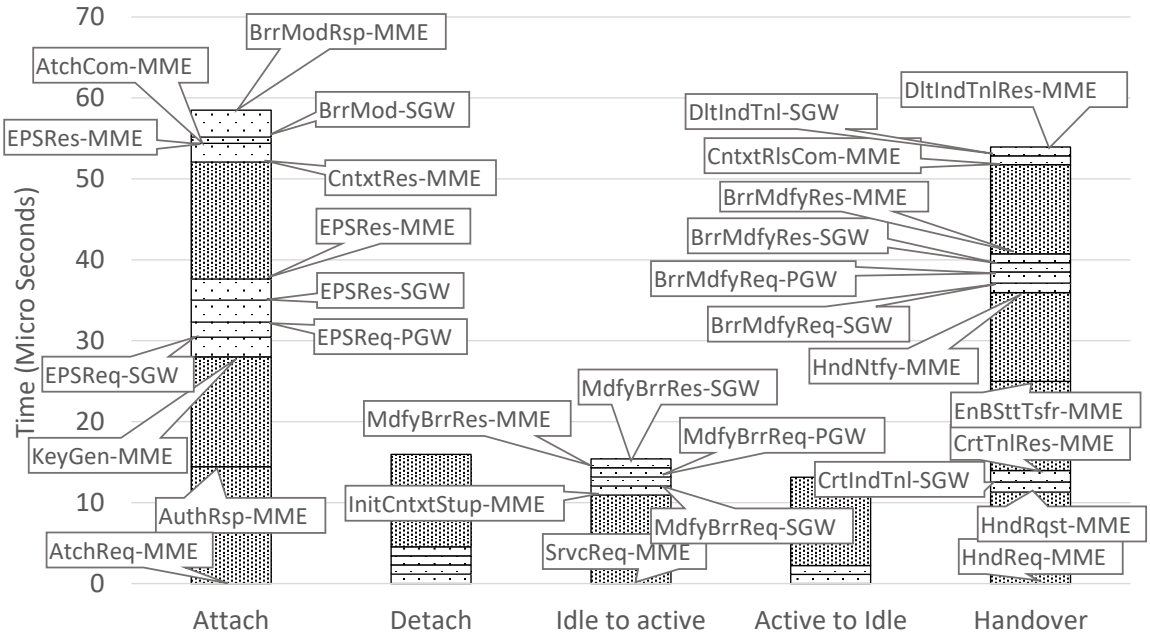


Figure 7.17: Detailed completion time for LTE EPC. Call-outs specify the receiver of that messages and where the timestamp is recorded

controller. Since the messages exchanged between the control and data plane components of the SGW and PGW are mediated by and forwarded through the SDN controller (e.g., using the OpenFlow protocol), there is a significant increase in latency (almost an order of magnitude). In the handover case, the three main chunks of time, numbered as 1,2 and 3, are bearer modification of PGW and SGW, and deleting the indirect tunnel on SGW (see Fig. 7.18). The second largest time spent is for messages between the control plane components and the eNodeB. The last category, which can be barely seen, is for messages exchanged between the components hosted in the same OpenNetVM host, that is between SGW, MME, PGW that are running on the same host. The reason for having this minimal delay is that we use shared memory packet buffers between these components to minimize the delay between them. This approach of having minimum delay was used to achieve the maximum possible fairness in comparison to our approach (in practice, they might be even implemented in separate machines which will increase the delay considerably regarding the delay between components).

7.7.6 Comparison with PEPC

One other work that we compare CleanG with, in terms of architecture, is [186] which has its own DPDK-based implementation. We have compared the performance of their system (PEPC) with CleanG and the result is shown in Fig. 7.19. (PEPC throughput numbers are based on the graph reported in [186] and the same workload and parameters are used for the CleanG experiment.) CleanG retains its high data plane throughput for large numbers of users and its performance degrades at a much lower rate. CleanG's control

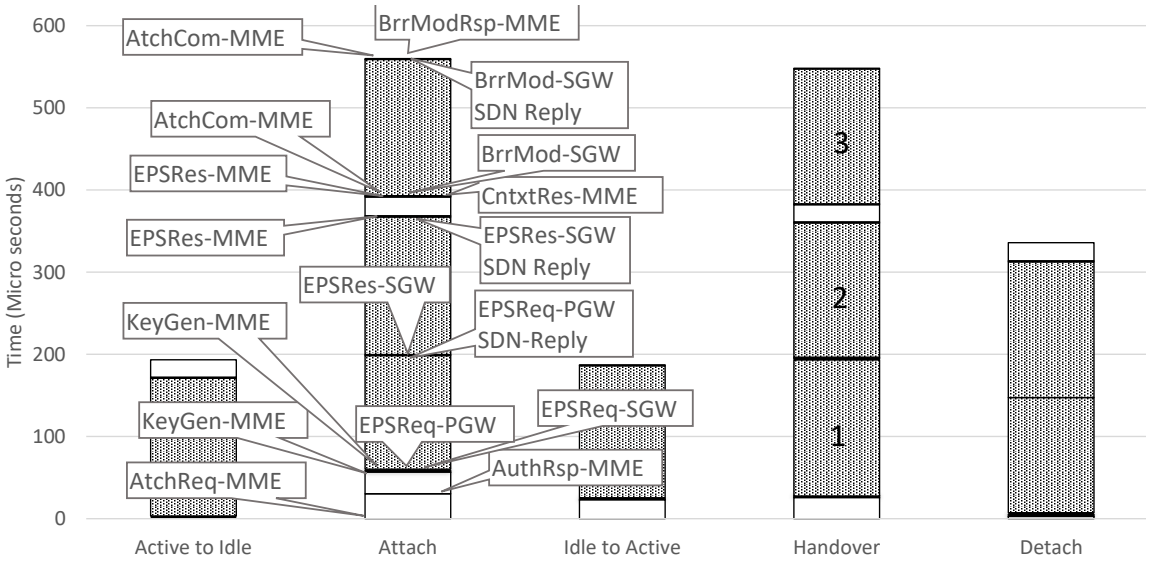


Figure 7.18: Detailed completion time of CUPS-based architecture

plane protocol is optimized in addition to the improvements arising from the architecture. CleanG drastically reduces the number of control messages exchanged compared to PEPC. Thus, the resources of cellular core can be focused on forwarding data plane messages. Additionally, the overhead of GTP tunneling is also removed (note that the control plane protocol of PEPC remains similar to the vanilla 3GPP LTE protocol). Eliminating the GTP tunnel setup avoids disrupting the data plane by control plane messages.

7.8 Related Work

In recent years, improving the cellular network has been the subject of numerous efforts both in research and industry [89, 196], especially as 5G is being deployed. However, there has been only limited focus on simplifying the cellular protocols. One work that has focused on the simplification of control plane is [135], which rightfully points out to the

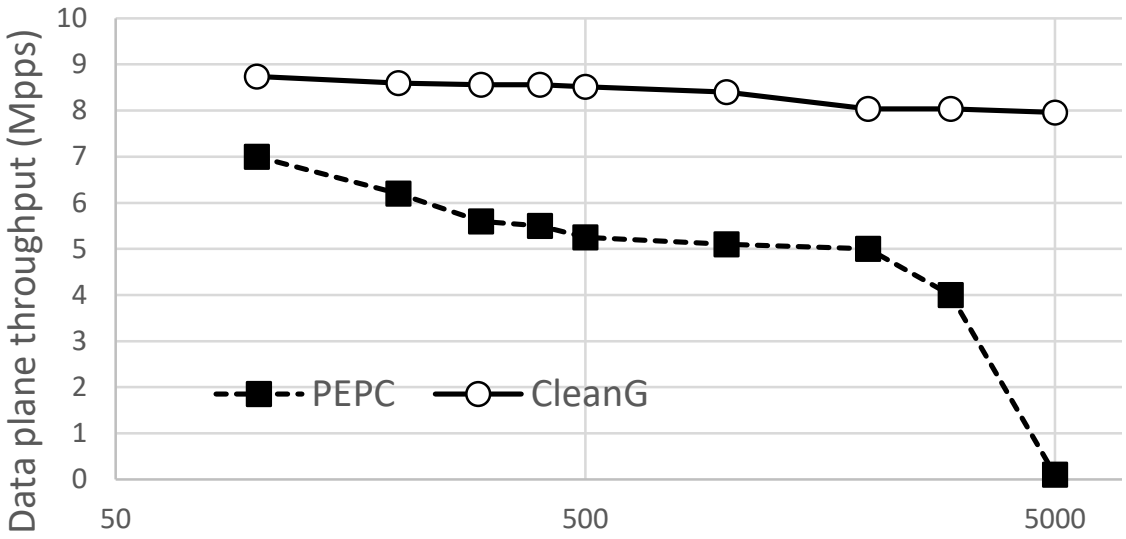


Figure 7.19: Maximum data plane rate based on the number of users in thousands

delays caused by control plane complexity and suggests improvements to it. However, we believe the potential of the optimization is not completely unlocked if possible improvements to the architecture and data plane forwarding are not also considered. A large body of recent research has focused on the separation of data and control plane and with and without the intervention of the SDN controller. [199] suggests using an SDN controller to shorten the path between UE's using P2P services. While this technique applies to the CleanG architecture as well, we observe that we can achieve similar or better performance by instantiating Core instances closer to the users. In [145] the EPC-edge is introduced as a termination for GTP tunnels, in addition to the separation. However, it still suffers from the complexity of the 3GPP protocol and architecture. Other work in the same vein include [11], and [120].

Another direction has been to introduce packet cores close to the edge of the network (e.g., at the telephony-related local central offices) [36]. CleanG can use a similar

approach for locating the CleanG Core Pools. Recently, [192] investigated the effects of unreliability in the virtualized core network and suggested using a proxy to mask these failures from the core message exchanges, as it can significantly hamper performance. Because of the use of a reliable underlying transport protocol and the fact that the number of messages is reduced and components are consolidated, we mitigate this effect in CleanG. Moreover, we can take advantage of reliability approaches for NFV platforms [207, 188].

A short motivation and introduction to CleanG's architecture were described in [159]. [186] follows the approach outlined in [159] for the design of the NFV-based cellular core by having the separation between the data and control NFs, with techniques to make the state tables more efficient. However, it retains the 3GPP protocol and suffers its inherent inefficiencies. Another closely related work is [184], which seeks to understand the bottlenecks in virtualizing cellular core network functions. Their observations are in sync with what we observed as well to motivate CleanG. Recently [47] has proposed an approach to use a streaming framework to implement specific cellular core components such as the MME efficiently, while retaining the 3GPP protocol. While the streaming framework is indeed useful, we believe a greater opportunity is also to see how the changes in both the architecture and protocol can be combined as in CleanG. Finally, techniques similar to [21] can be used to balance the load between different cores and pools in CleanG.

7.9 Conclusion

CleanG provides a simplified and scalable architecture and protocol for future cellular networks by intelligent adoption of NFV and SDN. We showed that its performance

is superior to several alternatives, across a number of scenarios. While we have considered the general scenario of the events and we have not addressed some corner cases such as roaming in this paper, we believe our design principles will be a valuable basis for developing a complete, production architecture and implementation for cellular networks going beyond 5G. The changes in the security aspect of the protocol are limited to how the necessary information is exchanged, and it should not affect the security algorithm and security properties of the protocol and architecture. Not only does CleanG improve the control plane, we also observed while it improves the scaling of the data plane by leveraging NFV. Further, we observe that the CUPS/SDN approach of separating the cellular control and data plane has significant drawbacks, because the tight coupling between the two is needed to continually mediate the flow of data packets in a cellular network. We will make the code for CleanG and our implementation of EPC and CUPS-based approaches open source on Github, to facilitate further research, evaluation and development work on the cellular network protocol and architecture.

Chapter 8

Our other efforts

So far, all the chapters were based on the works that I was the main author. In this chapter, we briefly cover other works that I helped the main author and I was not the main author.

8.1 SDNFV: Flexible and Dynamic Software Defined Control of an Application- and Flow-Aware Data Plan [239]

SDNFV is a framework to improve the task assignments in the software-based network. This framework divide the duties between the centralized controller, an agent on the host (called NF Manager), and each Network Function (NF). This framework is designed based on the zero-copy concept. Packets are exchanged between different components without copying the packet in memory. Just a pointer to the packet is added to the input ring buffer of different NFs and these NFs fetch the packets by using these buffers. NF manager uses several optimizations to reduce latency and improve the throughput. First,

if multiple NFs wants to read the packet, it improves the performance by letting multiple read happen in parallel. Hence, the NFs are not blocked by the processing of other NFs. Second, it caches the flow table look-up and store them in the packet descriptor. Third, it automatically balances the load between multiple instances of the NFs. As a part of this framework a placement engine is needed to assign different tasks to different NFs. For this purpose, a placement engine similar to the propose engine in Chapter 2 is used.

8.2 A scalable resource allocation scheme for NFV: Balancing utilization and path stretch [228]

This paper tackles a similar problem to the problem of Chapter 2. In this problem, in a network that uses software defined networking and networking function virtualization, in which each node is potentially able to run NFs and forward packets. Our algorithm tries to make a balance between flow level decision (routing) and network level decision (placement). First, a mixed integer linear programming formulation is proposed. The difference of this formulation with the formulation in the Chapter 2 is that this formulation maximizes the flow admission besides resource minimization proposed in that chapter. Hence, each flow can be admitted or not admitted to the network. Because of the limitation of scalability of this solution, a heuristic solution is suggested to solve this problem in larger scale. This heuristic tries to make a balance between the stretch in path and the utilization of network resources.

8.3 ClusPR: Balancing Multiple Objectives at Scale for NFV Resource Allocation [227]

ClusPR is the extention of the work we briefly explained in section 8.2. Besides improvements in the heuristic for the routing and placement, an online version of the heuristic is proposed in this paper as well. The online version of the algorithm is based on the offline version, but considers the assignments that are they are already made for the previous flows and does not change the placements that are already made.

Chapter 9

Conclusions

Software-based networks that use technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) while providing a large number of opportunities, they face new challenges as well. In this ultra-flexible and configurable environment, traditional solutions may not be completely effective anymore. In addition, the traditional separation between forwarding and processing nodes in the network is not valid anymore, and in the most general case, all the nodes can be considered as a forwarding or processing unit. In all different chapters of this thesis, we worked on the necessary changes in architecture, framework, and design of different algorithms to achieve the optimal solution for the software-based network. For example, traditionally placement and routing problems were solved separately. However, this approach does not lead to the best possible solution anymore. Because of this reason, in this thesis, not only we investigated the clear opportunities and challenges of software-based networks such as scalability, easy updating, and upgrading, but also we tried to look at the problem from a different angle than the

current common approach and seek the unique opportunities resulting from the specific characteristic of software-based networks. The result of this approach is achieving to more optimized network in comparison with using the current state-of-the-art architecture and frameworks and only implement their components in software.

In this thesis, we looked at these challenges and opportunities from different perspectives. At first, we looked at the problem resource assignment to different needs. In a software-based network, this problem translates to the routing and Network Function (NF) placement problem in two different levels. At first we solved the problem for the network level resource assignment, and then we solved the problem within a system that benefits from SmartNICs. Afterward, with the assumption that we have done the resource placement, and we have found the best route for the flows, we worked on a protocol to handle chains of services in this network. Finally, we redesigned architecture and protocol for the next generation of cellular networks that can benefit from the improved software-based network.

In the first branch of our work, the placement and routing in a software-based network, we proposed an algorithm that jointly considers both routing and placement of the network function. Because of joint consideration of the routing and placement, the output of this algorithm is more efficient in comparison with traditional approaches in which the placement and routing are optimized separately. Later, we worked on heuristics to achieve similar results faster and for the online version of the algorithm.

In the second branch of our work that we worked on the optimization within one system, we optimized the task assignment to the components within a system. Different

tables and actions of a P4 pipeline can be carried out by the host or by the SmartNIC. Our SDN agent makes an abstraction of the whole system as a single entity and provides an interface to the SDN controller and it optimizes the task assignment locally. Based on many different factors, such as data dependency, it assigns different tables and actions to the host and the SmartNIC.

The third branch of our work covered the protocol to enable efficient service chaining in the software-based network. Our proposed protocol reduces the need to add extra tags to packets to distinguished the step they are in their service chain. It also reduces the number of bytes needed to encode this information. It manages to achieve this optimization by leveraging the global knowledge stored in the centralized controller.

The fourth and the last branch of work focuses on applying our previous endeavors on the cellular network domain. To achieve an efficient cellular network, we claimed the necessity of designing an architecture and protocol considering the challenges and opportunities of software-based networks. Through a rigorous experiment and evaluation, we showed the improvement resulted from the simplified protocol and architecture, and the advantages of software-based environment.

We believe these steps are some of the first steps toward the true software-based network and still, lots of other opportunities and challenges exist in these networks. The more efficient heuristic that might be able to achieve solutions even closer to the optimal solution is imaginable for the joint routing and placement problem. Even, it might be possible to combine both optimization problems of network-wide and within a system as a one general optimization problem. While this combined problem, is not solvable based

on the capability of the current commercial solvers, the future generation of the solver and heuristics might be able to solve this problem. In the cellular network front, while we suggested the base for a simplified architecture and protocol, because of the sheer amount of details for all different scenarios, we could not cover all the corner cases. We hope other groups, such as standardization organizations follow our efforts in this path.

All in all, software-based networks provide a plethora of opportunities for the next generations of networks. However, to leverage them efficiently, we believe merely implementing hardware components as a software piece is not the answer, and it is necessary to rethink the networks and consider the specific challenges and opportunities of software-based networks.

Bibliography

[1] 3GPP. Signaling is growing 50 percent faster than data traffic. White paper 23.401, Nokia Siemens Networks, 2012. Version 15.3.0.

[2] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification. Technical Specification (TS) 36.321, 3rd Generation Partnership Project (3GPP), 2017. Version 15.1.0.

[3] 3GPP. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. Technical Specification (TS) 23.401, 3rd Generation Partnership Project (3GPP), 2017. Version 15.3.0.

[4] 3GPP. General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U). Technical Specification (TS) 29.281, 3rd Generation Partnership Project (3GPP), 2017. Version 15.2.0.

[5] 3gpp ts 23.502, procedures for the 5g system.

[6] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2211–2219. IEEE, 2013.

[7] Ian F. Akyildiza, Ahyoung Leea, Pu Wangb, Min Luoc, and Wu Chouc. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks (Elsevier)*, 2014.

[8] Mansoor Alicherry and TV Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. In *INFOCOM, 2013 Proceedings IEEE*, pages 647–655. IEEE, 2013.

[9] E. Allender, H. Buhrman, and M.Koucky. Power from random strings. *Annals of Pure and Applied Logic*, 2002.

[10] Nadav Amit, Muli Ben-Yehuda, and Ben-Ami Yassour. Iommu: strategies for mitigating the iotlb bottleneck. In *Proceedings of the 2010 international conference on Computer Architecture*, ISCA'10, pages 256–274, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] Xueli An, Wolfgang Kiess, and David Perez-Caparros. Virtualization of cellular network epc gateways based on a scalable sdn architecture. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2295–2301. IEEE, 2014.

[12] James W. Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. xomb: extensible open middleboxes with commodity servers. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ANCS '12, pages 49–60, New York, NY, USA, 2012. ACM.

[13] Bilal Anwer, Theophilus Benson, Nick Feamster, Dave Levin, and Jennifer Rexford. A slick control plane for network middleboxes. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 147–148, New York, NY, USA, 2013. ACM.

[14] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and Kadangode K Ramakrishnan. Optimal content placement for a large-scale vod system. In *Proceedings of the 6th International COnference*, page 4. ACM, 2010.

[15] Mayutan Arumaithurai, Jiachen Chen, Edo Monticelli, Xiaoming Fu, and Kadangode K. Ramakrishnan. Exploiting icn for flexible management of software-defined networks. In *Proceedings of the 1st International Conference on Information-centric Networking*, INC '14, pages 107–116, New York, NY, USA, 2014. ACM.

[16] Mayutan Arumaithurai, Jiachen Chen, Edo Monticelli, Xiaoming Fu, and Kadangode K. Ramakrishnan. Exploiting icn for flexible management of software-defined networks. In *Proceedings of the 1st International Conference on Information-centric Networking*, INC '14, pages 107–116, New York, NY, USA, 2014. ACM.

[17] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[18] Jordan Augé, Giovanna Carofiglio, Giulio Grassi, Luca Muscariello, Giovanni Pau, and Xuan Zeng. Anchor-less producer mobility in icn. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 189–190. ACM, 2015.

[19] D Awduche, Lou Berger, D Gan, Tony Li, Vijay Srinivasan, and George Swallow. Rfc 3209-rsvp-te: extensions to rsvp for lsp tunnels, 2001.

[20] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. Chatty tenants and the cloud network sharing problem. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 171–184, Berkeley, CA, USA, 2013. USENIX Association.

[21] Arijit Banerjee, Rajesh Mahindra, Karthik Sundaresan, Sneha Kasera, Kobus Van der Merwe, and Sampath Rangarajan. Scaling the lte control-plane for future mobile access. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 19:1–19:13, New York, NY, USA, 2015. ACM.

[22] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *In Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[23] Luiz Andre Barroso. Warehouse-scale computing: Entering the teenage decade. *SIGARCH Comput. Archit. News*, 39(3):–, June 2011.

[24] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[25] Muli Ben-Yehuda, Jimi Xenidis, Michal Ostrowski, Karl Rister, Alexis Bruemmer, and Leendert Van Doorn. The price of safety: Evaluating iommu performance. In *In Proceedings of the 2007 Linux Symposium*, 2007.

[26] Apache Bench. ab-apache http server benchmarking tool.

[27] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM.

[28] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM.

[29] Daniel G. Bobrow, Jerry D. Burchfiel, Daniel L. Murphy, and Raymond S. Tomlinson. Tenex, a paged time sharing system for the pdp - 10. *Commun. ACM*, 1972.

[30] Raffaele Bolla and Roberto Bruschi. Pc-based software routers: high performance and application service support. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '08, pages 27–32, New York, NY, USA, 2008. ACM.

[31] Pat Bosshart et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[32] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Proceedings of the 2010 IEEE 35th Conference on*

Local Computer Networks, LCN '10, pages 408–415, Washington, DC, USA, 2010. IEEE Computer Society.

[33] Mihai Budiu and Chris Dodd. The $p4_{16}$ programming language. *Operating Systems Review*, 51(1):5–14, 2017.

[34] Edouard Bugnion, Scott Devine Kinshuk Govil, and Mendel Rosenblum. Disco: Running commodity operating systems on sclable multiprocessors. *ACM Transactions on Computer Systems*, 1997.

[35] Ed. C. Filsfils, Ed. S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. Rfc 8402: Segment routing architecture. *IETF, July*, 2018.

[36] Ed. C. Perkins. M-Cord:Re-architecting Mobile Infrastructure to Enable 5G Networks. `https://www.opennetworking.org/solutions/m-cord/`. [Online; accessed 19-May-2018].

[37] Ed. C. Perkins. IP Mobility Support for IPv4, Revised. `https://tools.ietf.org/html/rfc5944`, November 2010.

[38] Carmelo Cascone. P4 support in onos. In *Proc. ONOS Build*, pages 1–29, 2017.

[39] Adrian Caulfield, Paolo Costa, and Monia Ghobadi. Beyond smartnics: Towards a fully programmable cloud. In *IEEE International Conference on High Performance Switching and Routing, ser. HPSR*, volume 18, 2018.

[40] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. Benchlab: an open testbed for realistic benchmarking of web applications. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.

[41] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. Benchlab: An open testbed for realistic benchmarking of web applications. *USENIX*, 2011.

[42] Cell phone tower statistics. http://www.statisticbrain.com/cell-phone-tower-statistics/.

[43] Cellular system support for ultra-low complexity and low throughput internet of things (ciot). http://www.3gpp.org/DynaReport/45820.htm.

[44] I. Cerrato, M. Annarumma, and F. Risso. Supporting fine-grained network functions through intel dpdk. *Proceedings of the 3rd IEEE European Workshop Software Defined Networks (EWSDN)*, 2014.

[45] Luwei Cheng and Cho-Li Wang. vbalance: using interrupt load balance to improve i/o performance for smp virtual machines. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 2:1–2:14, New York, NY, USA, 2012. ACM.

[46] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, September 2007.

[47] Junguk Cho and Jacobus Van der Merwe. Poster: A new scalable, programmable and evolvable mobile control plane platform. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, pages 540–542, New York, NY, USA, 2017. ACM.

[48] U. Chunduri, A. Clemm, and R. Li. Preferred path routing - a next-generation routing framework beyond segment routing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Dec 2018.

[49] U. Chunduri, R. Li, R. White, J. Tantsura, L. Contreras, and Y. Qu. Preferred path routing (ppr) in is-is. *IETF, June*, 2018.

[50] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20162021 White Paper. White paper, Cisco , 2017. Document ID:1454457600805266.

[51] Visual Networking Index Cisco. Global mobile data traffic forecast update, 2015–2020 white paper. *Document ID*, 958959758, 2016.

[52] Christopher Clark, Keir Fraser, Steven Hand, Jakob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. *NSDI*, 2005.

[53] Leandro C. Coelho. Linearization of the product of two variables. `http://www.leandro-coelho.com/linearization-product-variables/`, 2013. [Online; accessed 30-January-2015].

[54] Intel Corporation. Intel data plane development kit: Getting started guide, 2013.

[55] Data plane development kit. `http://dpdk.org/`, 2014.

[56] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 workshop on Experimental computer science*, ExpCS '07, New York, NY, USA, 2007. ACM.

[57] Jeff Dean. Designs, Lessons and Advice from Building Large Distributed Systems. LADIS Keynote, 2009.

[58] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 7–12, New York, NY, USA, 2013. ACM.

[59] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS*

22nd symposium on Operating systems principles, SOSP '09, pages 15–28, New York, NY, USA, 2009. ACM.

[60] Constantinos Dovrolis, Brad Thayer, and Parameswaran Ramanathan. Hip: Hybrid interrupt-polling for the network interface. *ACM Operating Systems Reviews*, 35:50–60, 2001.

[61] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iperf - the ultimate speed test tool for tcp, udp and sctp. `https://iperf.fr/`, 2014.

[62] Raphael Durner, Amir Varasteh, Max Stephan, Carmen Mas Machuca, and Wolfgang Kellerer. Hnlb: Utilizing hardware matching capabilities of nics for offloading stateful load balancers. *arXiv preprint arXiv:1902.03430*, 2019.

[63] T. Eckert, Y. Qu, and U. Chunduri. Preferred path routing (ppr) graphs - beyond signaling of paths to networks. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 384–390, Nov 2018.

[64] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. Moongen: a scriptable high-speed packet generator. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 275–287. ACM, 2015.

[65] A. Manzalini et al. Towards 5g software-defined ecosystems. http://sdn.ieee.org/images/files/pdf/towards-5g-software-defined-ecosystems.pdf.

[66] A. Rajan et al. Understanding the bottlenecks in virtualizing cellular core network functions. In *IEEE International Workshop on Local and Metropolitan Area Networks*, 2015.

[67] C. Marquezan et al. Identifying latency factors in sdn-based mobile core networks. In *ISCC*, 2016.

[68] P. Quinn et al. Network Service Header. https://tools.ietf.org/pdf/draft-quinn-sfc-nsh-07.pdf, 2015. [Online; accessed 12-October-2015].

[69] R. Sinha et al. Internet packet size distributions: Some observations. *USC/Information Sciences Institute, Tech. Rep.*, 2007.

[70] S. Palkar et al. E2: a framework for nfv applications. In *ACM SOSP*, 2015.

[71] Timothy Wood et al. Towards a software-based network: Integrating software defined networking and network function virtualizationa. *IEEE Network*, May-June 2015.

[72] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546, Berkeley, CA, USA, 2014. USENIX Association.

[73] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 19–24, New York, NY, USA, 2013. ACM.

[74] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn. *Queue*, 11(12):20:20–20:40, December 2013.

[75] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois. The segment routing architecture. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2015.

[76] Daniel Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation*, pages 315–328, 2017.

[77] Nate Foster, Michael J. Freedman, Rob Harrison, Jennifer Rexford, Matthew L. Meola, and David Walker. Frenetic: A high-level language for openflow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[78] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. Catalog of patterns of enterprise application architecture. *URL: http://martinfowler. com/eaaCatalog/index. html*, pages 25–26, 2003.

[79] Xiongzi Ge, Yi Liu, David H.C. Du, Liang Zhang, Hongguang Guan, Jian Chen, Yuping Zhao, and Xinyu Hu. Opennfv: Accelerating network function virtualization with a consolidated framework in openstack. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, New York, NY, USA, 2014. ACM.

[80] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual middleboxes as first-class entities. Technical report, Technical Report TR1771, University of Wisconsin-Madison, 2012.

[81] Aaron Gember, Anand Krishnamurthy, Saul St. John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Aditya Akella, and Vyas Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. *CoRR*, abs/1305.0209, 2013.

[82] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174, New York, NY, USA, 2014. ACM.

[83] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 163–174. ACM, 2014.

[84] Generic routing encapsulation (gre). https://tools.ietf.org/html/rfc2784.

[85] Mel Gorman. Understanding the linux virtual memory manager. *Prentice Hall*, 2004.

[86] Gprs enhancements for e-utran access. http://www.3gpp.org/ftp/Specs/htmlinfo/23401.htm.

[87] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.

[88] Adam Greenhalgh, Felipe Huici, Mickael Hoerdt, Panagiotis Papadimitriou, Mark Handley, and Laurent Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Comput. Commun. Rev.*, 39(2):20–26, March 2009.

[89] A. Gupta and R. K. Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE Access*, 3:1206–1232, 2015.

[90] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[91] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 357–371. ACM, 2018.

[92] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. *USENIX*, 2008.

[93] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, Feb 2015.

[94] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 195–206, New York, NY, USA, 2010. ACM.

[95] David Hancock and Jacobus Van der Merwe. Hyper4: Using p4 to virtualize the programmable data plane. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 35–49. ACM, 2016.

[96] M. He, A. Basta, A. Blenk, N. Deric, and W. Kellerer. P4nfv: An nfv architecture with flexible data plane reconfiguration. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 90–98, Nov 2018.

[97] Val Henson. An analysis of comprare-by-hash. *HotOS*, 2003.

[98] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 445–458, Berkeley, CA, USA, 2014. USENIX Association.

[99] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 445–458, Seattle, WA, 2014. USENIX Association.

[100] Jinho Hwang and Timothy Wood. Adaptive dynamic priority scheduling for virtual desktop infrastructures. *IEEE/ACM IWQoS*, 2012.

[101] Jinho Hwang and Timothy Wood. Adaptive performance-aware distributed memory caching. *USENIX Internation Conference on Autonomic Computing*, 2013.

[102] Cisco Visual Networking Index. Global mobile data traffic forecast update, 2012-2017. *Cisco white paper*, 2013.

[103] European Telecommunications Standards Institute. Network functions virtualisation (nfv)- introductory white paper. *White Paper*, 2012.

[104] European Telecommunications Standards Institute. Network functions virtualisation (nfv)- update white paper. *White Paper*, 2013.

[105] European Telecommunications Standards Institute. Network functions virtualisation (nfv). *White Paper*, 2014.

[106] European Telecommunications Standards Institute. Network functions virtualisation (nfv)- white paper 3. *White Paper*, 2014.

[107] European Telecommunications Standards Institute. Network functions virtualization (nfv): An introduction, benefits, enablers, challenges & call for action. *White Paper*, 2014.

[108] European Telecommunications Standards Institute. Network functions virtualization (nfv): Architectural framework. *White Paper*, 2014.

[109] European Telecommunications Standards Institute. Network functions virtualization (nfv): Use cases. *White Paper*, 2014.

[110] Intel. Supporting Evolved Packet Core for One Million Mobile Subscribers with Four IntelXeon Processor-Based Servers. Technical brief, Intel and Sprint, 2015. Document ID:1454457600805266.

[111] Intel Corp. Intel data plane development kit: Programmer's guide, 2019.

[112] Intel data direct i/o technology. http://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html.

[113] Muthurajan Jayakumar. Data plane development kit: Performance optimization guidelines, 2016.

[114] Predrag R. Jelenkovic and Ana Radovanovic. Optimizing lru caching for variable document sizes. *Combinatorics, Probability & Computing*, 13(4-5):627–643, 2004.

[115] Bob Jenkins. A hash function for hash table lookup.

[116] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 163–174, New York, NY, USA, 2013. ACM.

[117] Eric Jo, Deng Pan, Jason Liu, and Linda Butler. A simulation and emulation study of sdn-based multipath routing for fat-tree data center networks. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 3072–3083, Piscataway, NJ, USA, 2014. IEEE Press.

[118] Dilip A. Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 51–62, New York, NY, USA, 2008. ACM.

[119] S. Jouet, R. Cziva, and D. P. Pezaros. Arbitrary packet matching in openflow. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, July 2015.

[120] James Kempf, Bengt Johansson, Sten Pettersson, Harald Lüning, and Tord Nilsson. Moving the mobile evolved packet core to the cloud. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pages 784–791. IEEE, 2012.

[121] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM.

[122] Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, February 2013.

[123] Joongi Kim, Seonggu Huh, Keon Jang, KyoungSoo Park, and Sue Moon. The power of batching in the click modular router. In *Proceedings of the Asia-Pacific Workshop on Systems*, APSYS '12, pages 14:1–14:6, New York, NY, USA, 2012. ACM.

[124] Jacob Faber Kloster, Jesper Kristensen, and Arne Mejlholm. Efficient memory sharing in the xen virtual machine monitor. *Technical Report*, 2006.

[125] Younggyun Koh, Calton Pu, Sapan Bhatia, and Charles Consel. Efficient packet processing in userlevel os: A study of uml. In *in Proceedings of the 31th IEEE Conference on Local Computer Networks (LCN06)*, 2006.

[126] Eddie Kohler. The click modular router. *PhD Thesis*, 2000.

[127] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff,

Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Seattle, WA, April 2014. USENIX.

[128] Sameer G. Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K. K. Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 71–84, New York, NY, USA, 2017. ACM.

[129] Patrick Kutch and Brian Johnson. Sr-iov for nfv solutions. *Practical Considerations and Thoughts (Intel, Networking Division)*, 2017.

[130] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and Mahadev Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 1–12, New York, NY, USA, 2009. ACM.

[131] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M Swift, and TV Lakshman. Uno: uniflying host and smart nic offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 506–519. ACM, 2017.

[132] David Levinthal. Performance analysis guide for intel core i7 processor and intel xeon 5500, 2013.

[133] Chuanpeng Li, Chen Ding, and Kai Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 workshop on Experimental computer science*, ExpCS '07, New York, NY, USA, 2007. ACM.

[134] Yinan Li, Ippokratis Pandis, Rene Mueller, Vijayshankar Raman, and Guy Lohman. Numa-aware algorithms: the case of data shuffling. *The biennial Conference on Innovative Data Systems Research (CIDR)*, 2013.

[135] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. A control-plane perspective on reducing data access latency in lte networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, pages 56–69, New York, NY, USA, 2017. ACM.

[136] Jiuxing Liu and Bulent Abali. Virtualization polling engine (vpe): using dedicated cpu cores to accelerate i/o virtualization. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 225–234, New York, NY, USA, 2009. ACM.

[137] Robert Love. Linux kernel development. *Novell Press*, 2005.

[138] Frank Mademann. System architecture milestone of 5G Phase 1 is achieved. `http://www.3gpp.org/NEWS-EVENTS/3GPP-NEWS/1930-SYS_ARCHITECTURE`, December 21, 2017. [Online; accessed 19-May-2018].

[139] Dan Magenheimer, Chris Mason, Dave McCracken, and Kurt Hackel. Transcendent memory and linux. *Oracle Corp.*, 2009.

[140] Srihari Makineni, Ravi Iyer, Partha Sarangam, Donald Newell, Li Zhao, Ramesh Illikkal, and Jaideep Moses. Receive side coalescing for accelerating tcp/ip processing. In *Proceedings of the 13th International Conference on High Performance Computing*, HiPC'06, pages 289–300, Berlin, Heidelberg, 2006. Springer-Verlag.

[141] Aditya Agarwal Mark Slee and Marc Kwiatkowski. Abstract thrift: Scalable cross-language services implementation.

[142] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, April 2014. USENIX Association.

[143] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 459–473, Berkeley, CA, USA, 2014. USENIX Association.

[144] Miguel Masmano, I. Ripoll, Alfons Crespo, and Jorge Real. Tlsf: A new dynamic memory allocator for real-time systems. *ECRTS*, 2004.

[145] S Matsushima and R Wakikawa. Stateless user-plane architecture for virtualized epc (vepc). *draft-matsushima-stateless-uplane-vepc-04 (work in progress)*, 2015.

[146] N. McKeown. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[147] Nick McKeown et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38:69–74, 2008.

[148] Nick McKeown et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[149] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6, June 2014.

[150] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and T.V. Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 13–18, New York, NY, USA, 2014. ACM.

[151] Eliav Menachi and Ran Giladi. Hierarchical ethernet transport network architecture for backhaul cellular networks. *Wireless networks*, 19(8):1933–1943, 2013.

[152] Zili Meng, Jun Bi, Chen Sun, Shuhe Wang, Minhu Wang, and Hongxin Hu. Pam: When overloaded, push your neighbor aside! *arXiv preprint arXiv:1805.10434*, 2018.

[153] Grzegorz Milos, Derek G. Murray, Steven Hand, and Michael A. Fetterman. Satori: Enlightened page sharing. *USENIX*, 2009.

[154] Mobile radio interface signalling layer 3; general aspects. http://www.3gpp.org/dynareport/24007.htm.

[155] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15:217–252, 1997.

[156] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood. Virtual function placement and traffic steering in flexible and dynamic software defined networks. In *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*, pages 1–6, April 2015.

[157] A. Mohammadkhan and K. K. Ramakrishnan. Re-Architecting the Packet Core and Control Plane for Future Cellular Networks. In *The 27th IEEE International Conference on Network Protocols*, 2019.

[158] A. Mohammadkhan, K. K. Ramakrishnan, A. S. Rajan, and C. Maciocco. Considerations for re-designing the cellular infrastructure exploiting software-based networks. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, Nov 2016.

[159] Ali Mohammadkhan, K.K. Ramakrishnan, Ashok Sunder Rajan, and Christian Maciocco. Cleang: A clean-slate epc architecture and controlplane protocol for next generation cellular networks. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, CAN '16, pages 31–36, New York, NY, USA, 2016. ACM.

[160] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[161] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[162] Young Gyoun Moon, Ilwoo Park, Seungeon Lee, and Kyoung Soo Park. Accelerating flow processing middleboxes with programmable nics. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*, page 14. ACM, 2018.

[163] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Inc., 1st edition, 2016.

[164] S Neil, M Ratul, and A Thomas. Quantifying the causes of path inflation. In *Proc. ACM SIGCOMM'03*, 2003.

[165] NETMANIAS TECHNICAL DOCUMENTS. http://www.netmanias.com/.

[166] Netronome. Netronome nfp-4000 flow processor. `https://www.netronome.com/m/documents/PB-NFP-4000.pdf`.

[167] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. Openairinterface: A flexible platform for 5g research. *SIGCOMM Comput. Commun. Rev.*, 44(5):33–38, October 2014.

[168] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. *USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[169] Robert Olsson. Pktgen the linux packet generator. In *Proceedings of the Linux Symposium, Ottawa, Canada*, volume 2, pages 11–24, 2005.

[170] OpenFlow Switch Specification, Version 1.5.1. `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`. [Online; accessed 19-May-2018].

[171] Openflow switch specifications. `https://www.opennetworking.org/software-defined-standards/specifications`. Accessed:2019-05-01.

[172] $p4_{16}$ portable switch architecture(psa). `https://p4.org/p4-spec/docs/PSA.pdf`, 2019. [ONLINE].

[173] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[174] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of nfv. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, Berkeley, CA, USA, 2016. USENIX Association.

[175] VMWare White Paper. Vmware vnetwork distributed switch.

[176] Wind River White Paper. High-performance multi-core networking software design options, 2013.

[177] Edwin S Peer. High-speed and memory-efficient flow cache for network flow processors, February 2019. US Patent App. 15/356,562.

[178] Frank Yong Yang Peter Schmitt, Bruno Landais. Control and User Plane Separation of EPC nodes (CUPS). `http://www.3gpp.org/cups`, 2017. [Online; accessed 19-May-2018].

[179] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby. Virtual switching in an era of advanced edges. *Workshop on Data Center - Converged and Virtual Ethernet Switching (DC-CAVES)*, 2010.

[180] Ben Pfaff et al. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation*, pages 117–130, 2015.

[181] Ben Pfaff et al. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 117–130, 2015.

[182] S. Previdi, Ed. C. Filsfils, J. Leddy, S. Matsushima, and Ed. D. Voyer. Ipv6 segment routing header (srh). *IETF, March*, 2018.

[183] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu. Practical and incremental convergence between sdn and middleboxes. *ONS*, 2013.

[184] Zafar Ayyub Qazi, Phani Krishna Penumarthi, Vyas Sekar, Vijay Gopalakrishnan, Kaustubh Joshi, and Samir R. Das. Klein: A minimally disruptive design for an elastic cellular core. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 2:1–2:12, New York, NY, USA, 2016. ACM.

[185] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[186] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. A high performance packet core for next generation cellular networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 348–361, New York, NY, USA, 2017. ACM.

[187] Lin Quan and John Heidemann. On the characteristics and reasons of long-lived internet flows. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 444–450, New York, NY, USA, 2010. ACM.

[188] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 1:1–1:15, New York, NY, USA, 2013. ACM.

[189] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 227–240, Lombard, IL, 2013. USENIX.

[190] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: system support for elastic execution in virtual middleboxes. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 227–240, Berkeley, CA, USA, 2013. USENIX Association.

[191] Kaushik Kumar Ram, Alan L. Cox, Mehul Chadha, and Scott Rixner. Hyper-switch: A scalable software virtual switching architecture. *USENIX Annual Technical Conference (USENIX ATC)*, 2013.

[192] M. T. Raza, D. Kim, K. H. Kim, S. Lu, and M. Gerla. Rethinking lte network functions virtualization. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, Oct 2017.

[193] Luigi Rizzo. netmap: A novel framework for fast packet I/O. In *USENIX Annual Technical Conference*, pages 101–112, Berkeley, CA, 2012. USENIX.

[194] Luigi Rizzo, Giuseppe Lettieri, and Vincenzo Maffione. Speeding up packet i/o in virtual machines. In *Architectures for Networking and Communications Systems (ANCS), 2013 ACM/IEEE Symposium on*, pages 47–58, Oct 2013.

[195] Eric Rosen, Arun Viswanathan, and Ross Callon. Rfc 3031: Multiprotocol label switching architecture. *IETF, January*, 2001.

[196] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi. Mobile network architecture evolution toward 5g. *IEEE Communications Magazine*, 54(5):84–91, May 2016.

[197] Jose Renato Santos, Yoshio Turner, G. Janakiraman, and Ian Pratt. Bridging the gap between software and hardware techniques for i/o virtualization. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[198] V. Sathiyamoorthi and V. Murali Bhaskaran. Web caching through modified cache replacement algorithm. *ICRTIT*, 2012.

[199] Ryan Saunders, Junguk Cho, Arijit Banerjee, Frederico Rocha, and Jacobus Van der Merwe. P2p offloading in mobile networks using sdn. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 3:1–3:7, New York, NY, USA, 2016. ACM.

[200] Karen Scarfone and Paul Hoffman. Guidelines on firewalls and firewall policy. *National Institute of Standards and Technology*, 2009.

[201] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[202] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 24–24. USENIX Association, 2012.

[203] Ivan Seskar, Kiran Nagaraja, Sam Nelson, and Dipankar Raychaudhuri. Mobility-first future internet architecture project. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pages 1–3, New York, NY, USA, 2011. ACM.

[204] Muhammad Zubair Shafiq, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. A first look at cellular machine-to-machine traffic: Large scale measurement and characterization. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 65–76, New York, NY, USA, 2012. ACM.

[205] R. Shah, M. Vutukuru, and P. Kulkarni. Cuttlefish: Hierarchical sdn controllers with adaptive offload. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 198–208, Sep. 2018.

[206] J. Sherry and S Ratnasamy. A survey of enterprise midlebox deployments. Technical report, Technical Report No. UCB/EECS-2012-24, 2012.

[207] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. Rollback-recovery for middleboxes. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 227–240, New York, NY, USA, 2015. ACM.

[208] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, CCS '13, pages 413–424, New York, NY, USA, 2013. ACM.

[209] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, 2008.

[210] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, April 2011.

[211] Christopher Stewart, Aniket Chakrabarti, and Rean Griffith. Zoolander: Efficiently meeting very strict, low-latency slos. *USENIX Internation Conference on Autonomic Computing*, 2013.

[212] Study on architecture for next generation system.

[213] System architecture for the 5g system (5gs).

[214] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94, October 2007.

[215] Guan-Hua Tu, Yuanjie Li, Chunyi Peng, Chi-Yu Li, Hongyi Wang, and Songwu Lu. Control-plane protocol interactions in cellular networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 223–234, New York, NY, USA, 2014. ACM.

[216] Ed. U. Chunduri, R. Li, J. Tantsura, L. Contreras, and X. De Foy. Transport network aware mobility for 5g. *IETF, July*, 2018.

[217] Erik-Jan van Baaren. Wikibench: A distributed, wikipedia based web application benchmark. *Master Thesis*, 2009.

[218] Steve VanDeBogart, Christopher Frost, and Eddie Kohler. Reducing seek overhead with application-directed prefetching. *USENIX*, 2009.

[219] Virtual extensible local area network (vxlan). https://tools.ietf.org/html/rfc7348.

[220] VMware. Resource management with vmware drs. *Technical Resource Center*, 2006.

[221] Andreas Voellmy, Ashish Agarwal, and Paul Hudak. Nettle: Functional reactive programming for openflow networks. Technical report, DTIC Document, 2010.

[222] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *In Proceedings of the ACM Symposium on Operating Systems Principles*, 2005.

[223] Open vSwitch. http://www.openvswitch.org.

[224] Carl A. Waldspurger. Memory resource management in vmware esx server. *OSDI*, 2002.

[225] Liufeng Wang, Huaimin Wang, Lu Cai, Rui Chu, Pengfei Zhang, and Lanzheng Liu. A hierarchical memory service mechanism in server consolidation environment. *ICPADS*, 2011.

[226] Dan Williams, Hani Jamjoom, Yew-Huey Liu, and Hakim Weatherspoon. Overdriver: handling memory overload in an oversubscribed cloud. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 205–216, New York, NY, USA, 2011. ACM.

[227] Y. T. Woldeyohannes, A. Mohammadkhan, K. K. Ramakrishnan, and Y. Jiang. Cluspr: Balancing multiple objectives at scale for nfv resource allocation. *IEEE Transactions on Network and Service Management*, 15(4):1307–1321, Dec 2018.

[228] Y. T. Woldeyohannes, A. Mohammadkhan, K. K. Ramakrishnan, and Y. Jiang. A scalable resource allocation scheme for nfv: Balancing utilization and path stretch. In *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 1–8, Feb 2018.

[229] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang. Toward a software-based network: integrating software defined networking and network function virtualization. *IEEE Network*, 29(3):36–41, May 2015.

[230] Wenji Wu, Matt Crawford, and Mark Bowden. The performance analysis of linux networking - packet receiving. *Comput. Commun.*, 30(5):1044–1057, March 2007.

[231] Cong Xu, Sahan Gamage, Hui Lu, Ramana Kompella, and Dongyan Xu. vturbo: Accelerating virtual machine i/o processing using designated turbo-sliced core. *USENIX Annual Technical Conference*, 2013.

[232] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. T-storm: Traffic-aware online scheduling in storm. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 535–544. IEEE, 2014.

[233] Jisoo Yang, Dave B. Minturn, and Frank Hady. When poll is better than interrupt. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 3–3, Berkeley, CA, USA, 2012. USENIX Association.

[234] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 29–42, Berkeley, CA, USA, 2013. USENIX Association.

[235] Frank Yue. Network functions virtualization - everything old is new again. *http://www.f5.com/pdf/white-papers/service-provider-nfv-white-paper.pdf*, 2013.

[236] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, and J. Wu. Hyperv: A high performance hypervisor for virtualization of the programmable data plane. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, July 2017.

[237] C. Zhang, J. Bi, Y. Zhou, and J. Wu. Hypervdp: High-performance virtualization of the programmable data plane. *IEEE Journal on Selected Areas in Communications*, 37(3):556–569, March 2019.

[238] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K.K. Ramakrishnan, and Timothy Wood. Flurries: Countless fine-grained nfs for flexible per-flow customization. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 3–17, New York, NY, USA, 2016. ACM.

[239] Wei Zhang, Guyue Liu, Ali Mohammadkhan, Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Sdnfv: Flexible and dynamic software defined control of an application- and flow-aware data plane. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 2:1–2:12, New York, NY, USA, 2016. ACM.

[240] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMIddlebox '16, pages 26–31, New York, NY, USA, 2016. ACM.

[241] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMIddlebox '16, pages 26–31, New York, NY, USA, 2016. ACM.

[242] Wei Zhang, Timothy Wood, K.K. Ramakrishnan, and Jinho Hwang. Smartswitch: Blurring the line between network infrastructure & cloud applications. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, PA, June 2014. USENIX Association.

[243] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patney, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. Steering: A software-defined networking for inline service chaining. In *ICNP*, pages 1–10, 2013.

[244] Weiming Zhao and Zhenlin Wang. Dynamic memory balancing for virtual machines. *VEE*, 2009.

[245] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Saving cache by using less cache. *HotCloud*, 2012.