OPTIMIZATION OF NETWORK RESOURCE ALLOCATION FOR SOFTWARE-DEFINED DATA CENTER NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Chuanji Zhang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2019

OPTIMIZATION OF NETWORK RESOURCE ALLOCATION FOR SOFTWARE-DEFINED DATA CENTER NETWORKS

Approved by:

Dr. Douglas M. Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Henry L. Owen III
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yusun Chang
School of Electrical Engineering
Kennesaw State University

Dr. Linda M. Wills
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ellen W. Zegura
School of Computer Science
Georgia Institute of Technology

Date Approved: July 30, 2019

ACKNOWLEDGEMENTS

Over the past five years of my doctoral study, I have received support and help from a great number of individuals in different forms. First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Douglas M. Blough, for his continuous encouragement and support during my doctoral study. His excellent guidance and intellectual insights help me steadily advance towards the successful completion of my Ph.D. program, as well as this dissertation. It has been an honor and privilege to be his student and work with him. While thanking Dr. Blough, I must also thank Dr. George Riley. I am a better person and researcher for knowing him. Although he is not with us anymore, I will always remember his guidance and encouragements.

I would also like to thank all the academic members of the School of Electrical and Computer Engineering at the Georgia Institute of Technology for their help throughout my Ph.D. program. Special thanks go to Dr. Henry Owen and Dr. Yusun Chang, who provided me valuable and constructive suggestions on my work and also kindly serve in my Ph.D. Defense Reading Committee. I also thank Dr. Linda Wills and Dr. Ellen Zegura who kindly serve in my Ph.D. Defense Committee. All of their valuable comments and advice have greatly improved my research and the quality of this dissertation.

I would also like to thank my friends and colleagues at Georgia Tech, Hemin Yang, Mengyao Ge, Qiang Hu, Yan Yan, and Yuchen Liu, for their everlasting friendship and constant support, and for every unforgettable moment we have had together at Atlanta.

Finally, I would like to express my gratitude to my family, particularly to my parents, for their endless love and support. I could never successfully complete my Ph.D. program without your trust, sacrifice, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As cloud computing and data center network flourishes, the network that was once designed to support traditional networking scenario must now satisfy new requirements, such as on-demand dynamic connectivity, central management, agility, scalability, etc, to suit for the cloud environment and increasing demands. Improvements in networking technology are expected to help accommodate the new requirements. In particular, the Software-Defined Networking (SDN) paradigm, with the control plane separated from the data plane, is widely regarded as the next-generation networking technique.

The objective of this work is to optimize network resources allocation in the software-defined data center networks (DCN). The SDN resources considered here are the SDN switch to controller link bandwidth and the switch flow table size. First, in view of the proposal of utilizing multiple connections between switches and controller by OpenFlow 1.3, a queueing model is formulated accordingly. The model considers different interactions between switches and controller and can be used to provision the network with an appropriate number of switch-to-controller connections. Second, a controller-level admission control mechanism is proposed to determine if a new flow should be admitted to the network. The mechanism utilizes only the flow information and statistics collected by existing controller functionalities. With tolerable overheads, it strives to provide the best service quality to most users and at the same time generate good benefits for the service providers. Third, we study the fair and high-satisfaction resources allocation problem with the routing path optimized in software-defined DCN. We aggregate individual flows into flow groups and then find the optimal routing paths and the corresponding resource allocation vectors for each flow group. Three fairness models with a relaxation parameter are considered. Fourth, with a new formulation of the end-to-end delay in software-defined DCN derived using queueing theory, a new resources allocation problem which considers both the fairness constraint and the delay constraint is studied. Fifth, some practical issues, including the algorithm

complexity, the rounding effects, and the algorithm sensitivity to varying traffic demands, are considered for the resources allocation algorithms. The provided theoretical analysis and simulation results in this dissertation improve the efficiency of resource allocation in software-defined DCN.

CHAPTER 1
INTRODUCTION

Over the last decade, data center network (DCN) has experienced dramatic growth in terms of scale and traffic, and will reach the zettabyte era in the near future. Although technologies like SSD storage and virtualization are expected to help DCN to accommodate the huge traffic demands, easy policy update and application optimization, efficient resource allocation, and dynamically routing will be required in future DCN. In order to fulfill these requirements, DCN providers have turned to software-defined DCN for help in view of the benefits brought by software-defined networking (SDN).

1.1 Motivation and Research Objectives

SDN enables flexible network management and fast network innovations. Dynamic and optimized management of network resources becomes possible with the controller's global view. However, the resource allocation problem in software-defined DCN exposes itself to new challenges in this scenario. As the focus of this dissertation, one of the main challenges is to cope with the new resource constraints brought by the unique architecture of SDN. That is the capacity of the switch-to-controller link, which limits the message exchange rate between switch and controller, and the size of the flow table, which restricts the number of flow entries support by the switch. Additional research challenges range from enabling multi-tenant resource allocation and network resource utilization monitoring, etc. Next, we review the two resource limitations in details.

- The flow table of most commodity OpenFlow switches are placed in Ternary Content Addressable Memory (TCAM) to seek for single-clock-cycle lookup time. The size of the TCAM is limited due to power, cost, and chip size constraints and thus

the flow table size in OpenFlow switches is limited in real physical implementation. For example, the HP 5406zl switch hardware can only support 1500 flow entries [1] and the Broadcom chipset, which is widely used in commercial switches, can only support 2000 flow entries [2]. More recent hardware OpenFlow switch products can support up to 16000 [3] flow entries with the cost of high price and power [4, 5]. However, in a normal data center, the arrival rate of flows can reach 10000 flows per second per server rack [6]. Moreover, SDN allows fine-grained control of traffic. Instead of using the traditional 5-tuple definition of flows, OpenFlow 1.5.0 [7] defined 45 fields to identify a flow. The involved fields include VLAN ID, TCP flags, etc. This fine-grained control would dramatically increase the number of flows need to be managed in the network compared with the traditional 5-tuple definition and consequently make the situation worse.

- In terms of the switch-to-controller link bandwidth, the measured loopback bandwidth between the Application Specific Integrated Circuit (ASIC) and the management CPU of an OpenFlow switch is 80 Mbps in the HP 5406zl switch [1]. The switch only supports limited control channel bandwidth due to its low internal control path bandwidth and limited CPU performance. However, in a normal data center with 100 edge switches, the controller can see as many as 10 million new flows per second in the worst case [6]. Assuming the average size of the packets sent by the switch to the controller for each new flow is 100 Bytes, the required switch-to-controller link bandwidth per edge switch is at least 80 Mbps. The demand will get much higher on the core switches and aggregation switches, in a commercial cloud data center, and as the traffic demands in DCN increases. Besides, the collection of statistics and network statuses, such as the traffic rate in each link and the available flow table size, is also conducted via the switch-to-controller link. Such communications contribute to the traffic load on the switch-to-controller link by a non-negligible fraction [8]. Moreover, in-band SDN is proposed [9] and reinforces the need to have

sufficient switch-to-controller link bandwidth.

The large gaps between the demands and the available resources will cause packet loss and intolerable latency in the network. Solutions need to be investigated to address these limitations, and at the same time, the resources allocation algorithms designed for software-defined DCN should take these new constraints into consideration. The objective of this work is to optimize network resource allocation for software-defined DCN, constrained by the aforementioned capacity limitations.

1.2 Contributions and Scope

The primary contributions of this work are:

- The first contribution (Chapter 3) is that we develop a queueing model for auxiliary-connection-enabled SDN switches with limited buffer space. The model captures not only the *packet_in* interaction, but also the *stats/feature_request/reply* interactions, which are of great importance to enable SDN benefits, such as efficient resource allocation. The model is validated for lower traffic loads using the network simulator ns-3 [10], where network protocols including OpenFlow are implemented. Our model is then used to perform an in-depth analysis of switch performance under realistic data center traffic loads. With limited hardware resource available at a switch to build auxiliary connections, our model can serve as a tool to predict network performance and provision switches with the appropriate number of connections to satisfy the network quality of service (QoS) requirements.

- The second contribution (Chapter 4) is that we propose and analyze a lightweight admission control mechanism for SDN, which controls the flow entrance to the network when the flow table is congested. The admission control mechanism is implemented at the controller level and strives to provide the best QoS to most users and generate good benefits for service providers. Simulations are carried out using a canonical

three-tier data center topology and the mechanism is thoroughly evaluated in terms of data plane performance and control plane performance.

- The third contribution (Chapter 5) is that we address the resource allocation problem in software-defined DCNs, with the objective to maximize the total satisfaction ratio of the two SDN resources, subject to three different fairness constraints. A relaxation parameter $\delta \in [0, 1]$ is also introduced into these fairness models to allow the network operator to control the trade-off between the total demand satisfaction and fairness. Our approach is to aggregate individual flows into flow groups and then find the optimal routing path and the corresponding resource allocation vectors for each flow group satisfying all constraints.

- The fourth contribution (Chapter 6) is that we strive to provide delay guarantees to the delay-sensitive flows when allocating the SDN resources as in our previous work. The end-to-end delay constraint for each packet, considering both the control plane delay and data plane delay in SDN, is derived using queueing theory. To the best of our knowledge, this is the first work to provide a solution to the resource-constrained delay-guarantee problem considering the unique architecture of SDN.

- The fifth contribution (Chapter 7) is that we study some practical considerations for the proposed resource allocation algorithms to be implemented in the real world. The algorithm complexity and running time, the rounding effects, and its sensitivity to varying traffic demands are studied. The results confirm that our algorithms are scalable and can perform stably in a regular-size DCN with varying traffic demands.

The queueing model for auxiliary-connection-enabled SDN switches in Chapter 3 is developed for the single switch case, because the goal is to appropriately provision individual switches. The admission control mechanism studied in Chapter 4 targets a single controller SDN domain. The resource allocation algorithms studied in Chapter 5, 6, and 7 are designed for small and medium sized DCNs. As shown in Chapter 7, the optimization

problem for DCNs with 50-60 switches can be solved in a few minutes. Unfortunately, due to the exponential growth in running time, use of the algorithms for very large DCNs, e.g. 100 or more switches, is not feasible with current processing hardware.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we provide a brief introduction of basic SDN concepts and OpenFlow protocol, and discuss the related works. Then, in Chapter 3, the switch-to-controller link capacity is investigated by developing a queueing model of auxiliary-connection-enabled SDN switches. Moreover, in Chapter 4, an admission control mechanism in software-defined DCN is proposed to control the flow entrance when the flow table is congested. In addition, the problem of high satisfaction and fair allocation of resources in software-defined DCNs, including routing path selection, is addressed in Chapter 5. Different types of fairness models are studied. In Chapter 6, the delay guarantee for delay-sensitive flows in SDN is provided based on the work in Chapter 5. In Chapter 7, we study some of the practical issues for the resource allocation algorithms to be implemented in real world. Finally, in Chapter 8, our conclusions and suggestions for future work are provided.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Cloud Computing and Data Center Network

With rapid advances of technologies in processing and storage and the success of the Internet, computing resources have become cheaper, more powerful and more ubiquitously available. These technological trends have enabled the realization of cloud computing, where different kinds of computing services, including servers, storage, databases, networking, software, analytics, and intelligence, are delivered over the Internet ("the cloud"). Most cloud computing services fall into three broad categories: infrastructure as a service (IaaS), platform as a service (Paas), and software as a service (SaaS). Cloud computing is a critical shift from the traditional way companies think about IT Resources. It provides faster innovation, flexible resources and economies of scale. Over the last few years, cloud adoption has evolved from a developing technology to a well-established networking solution that is widely accepted and deployed. Google, Amazon, Microsoft, and other industrial giants have heavily invested in cloud computing and products like Google Drive, Amazon AWS and Microsoft Azure are making great profits to the companies. The widespread use of cloud computing is accelerating traffic growth, changing the end-user experience, and bringing new requirements and demands to data centers and cloud-based infrastructures.

A data center interconnects components such as servers, communication media, and data storage facilities and thus enables the usage of cloud computing. DCN plays a key role in a data center because it incorporates all of the data center resources together. DCN needs to be scalable and efficient to connect tens or even hundreds of thousands of servers to meet the growing demands of Cloud computing. Commonly used DCN topology includes three-tier DCN, fat tree DCN, etc. In both the three-tier DCN and the fat tree DCN,

(a) Global IP traffic in data centers [Zettabytes]

(b) Amount of data stored in data centers [Exabytes]

Figure 2.1: Data center usage in 2016 - 2021 according to [11]

the network topology is composed of three layers of network switches, namely the edge layer, the aggregation layer, and the core layer. The servers are connected directly to the edge layer switches. The aggregation layer switches are interconnected via the core layer switches. The core layer switches are also responsible for connecting the data center to other data centers and to the Internet. Over the last decade, DCN has experienced dramatic growth in terms of scale and traffic. According to the Cisco global cloud index [11] as shown in Fig. 2.1, the traffic demand and the data storage in DCN are experiencing a 20%-30% compound annual growth rate now. We are already in the zettabyte era. Although technologies like SSD storage and virtualization are expected to help DCN to accommodate the huge traffic demands, more and more network capabilities are required to cope with the cloud environment. Some of the desired network capabilities are summarized as follows.

- On-demand and dynamic connectivity: dynamically allocate the network resources

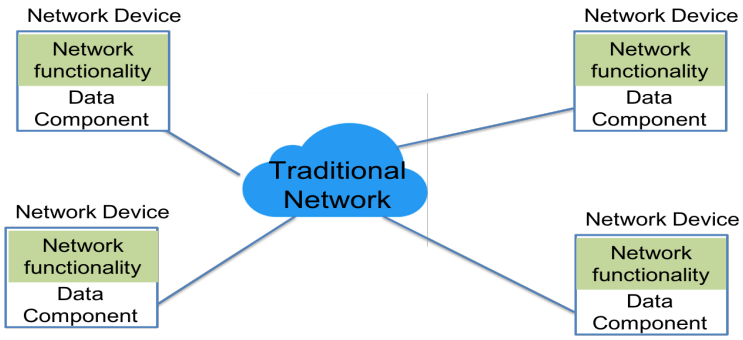
and build network connections to meet the evolving needs of services.

- Acceleration: speed up the deployment of workloads in a non-disruptive manner.
- Security: prevent security vulnerabilities from spreading across the network and deploy new mitigation policies easily.
- Central management: define and control policies that govern both physical and virtual networks in a central-controlled manner.
- Agility: implement and update network policies fast and consistently at scale.
- Scalability, programmability, automation, etc.

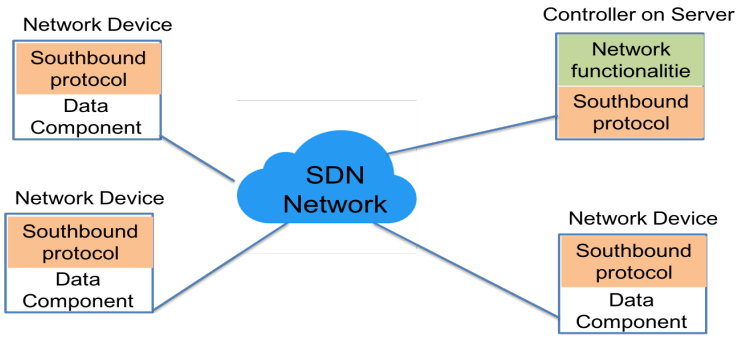
In order to fulfill these requirements, DCN providers have turned to SDN for help. For example, A commercial deployment of SDN in clouds was proposed by Nicira in [12]. Google leveraged SDN principles to build its Jupiter network which achieves a capacity scale-up by 100 times [13]. Besides the usage in clouds and DCN, SDN has also been invested in wide-area-network (WAN). In 2013, Microsoft presented its software-driven WAN (SWAN) [14], and according to the data-driven simulations of two production networks, the SWAN carries 60% more traffic than the current practice. Afterward, NTT launched software-defined WAN which spans over 190 countries in 2017 [15]. In the following section, we will elaborate on the concepts, history, and benefits of SDN.

2.2 SDN and OpenFlow Protocol

As shown in Fig. 2.2a, in traditional network, the networking functions are mainly implemented on dedicated devices, such as switches, routers, etc. Besides, most of the functionalities in these dedicated devices are implemented in dedicated hardware, such as ASIC. In other words, the network devices in traditional network are closed and bundling with both software and hardware. The control component and data component are vertically integrated in this case. This hardware-centric paradigm leads to many limitations.



(a) Traditional network architecture



(b) SDN architecture

Figure 2.2: Traditional network vs. SDN

- Vendor-locked devices: some of the network devices come from the vendors as black boxes. The control logic is already embedded in the device when shipped. Any changes of the control logic would be complex and cumbersome. Moreover, a network administrator with extensive knowledge of all device types is required under this situation.

- Time consuming and error-prone network configuration: since the network functionalities are implemented on individual devices and coupled with hardware, manual configurations are required to update logic functions, add devices, and remove devices. This static configuration approach makes it very complex to deploy a consistent set of policies. Security breaches, non-compliance, and inconsistent configurations are likely to happen. It fails to meet the demands of the continuously changing network.

- Inefficient network management: Since the network devices are distributed and operated according to specific protocols, the control and management of the numerous and heterogeneous networking devices would be inefficient. Besides, the balance between over-provision and under-provision is difficult to maintain due to a lack of a global view.

- Distributed network control: with the usage of various types of applications growing, the network needs to accommodate different levels of QoS/QoE requirements of these applications, such as the dynamic multimedia streams, Internet of Things, etc. Thus, complex and high-level policies taking care of different network events are needed to deliver the required QoS/QoE. However, the highly constrained low-level device configuration commands are not sufficient to fulfill the task. Moreover, a complex policy usually involves the corporation of multiple network devices, which requires tedious manual configuration on each device as stated before.

2.2.1 SDN

In order to tackle the aforementioned limitations and suit the new cloud environment, the Open Networking Foundation (ONF) [16], a user-led organization dedicated to the promotion and adoption of SDN, proposed a novel network architecture, SDN, where the control plane and data plane are separated with each other as shown in Fig. 2.2b. The control plane and data plane can evolve independently since they are not integrated anymore. The history of SDN [17] can be traced back to the mid- 1990s, when the active networks were proposed to introduce programmable functions in the network to enable greater innovations [18, 19]. Before the proposal of active network, researchers usually designed and tested new network protocols in small lab settings, simulated larger network behaviors, and then asked the Internet Engineering Task Force (IETF) [20] to finalize the protocols as standards. As the Internet took off in early- to mid- 1990s, the conventional way to test and deploy new ideas for improving network services is slow and cumbersome so researchers were looking for al-

ternative approaches. This was when active networking came to the table. The subsequent efforts were narrowly focused on routing and configuration management. These efforts drew a clear distinction and separation between the control functions and data planes [17, 21]. The history of the control plane and data plane separation can actually be traced back to the public switched telephone network, when people tried to separate the control and data plane as a way to simplify provisioning and management. This separation made it possible to solely focus on innovations in the control plane. The IETF also published a few standards from the year 2004 on decoupling the control and forwarding functions [22, 23]. However, these attempts failed to gain much attention at that time because the Internet community saw this separation as a risky action. The control plane is a single point failure in the network and the application programming interfaces (APIs) between the control and data plane would cause increasing competition. Finally, the emergence of OpenFlow API [24] and controller platforms such as NOX [25] (from 2007 to around 2010), represented the first instance of widespread acceptance of an open interface and enabled scalable and practical control-data plane separation.

The SDN framework (as shown in Fig. 2.3) consists of an application layer, a control layer, an infrastructure layer, and the interfaces between them. The controller lays on the control layer and is the ”brain” of the network. It communicates with business applications in the application layer through the northbound APIs, and relays the information to the network infrastructure, like switches, routers, etc, through the southbound protocols. In SDN, the controller has a full control over the whole network. It configures the network behaviors by installing flow entries on the switches. With this unique architecture, SDN enables many benefits which cannot be achieved using traditional networking technologies. Those benefits are summarized as follows.

- Programmability: first, the northbound APIs enable the application developers to program the network directly. Any choosing control policies can be implemented with a small cost. Network function implementation and update can be automated

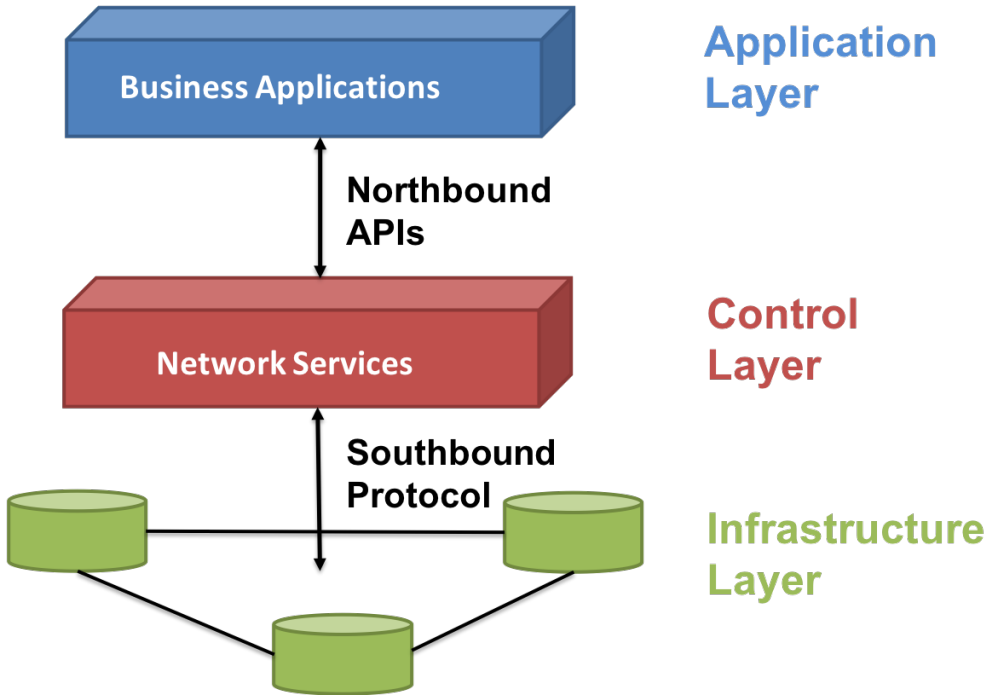


Figure 2.3: SDN framework

since they are in the form of high-level programming codes. In traditional network, this can only be done through protocols and tedious manual device configuration.

- Efficient network management: second, with the controller's global view, the network applications can be more intelligent and proficient to handle different network scenarios. For example, flexible resource management, optimized traffic engineering policies, and better mitigation mechanism based on current traffic status in the network can be achieved.

- Centralization: third, SDN enables central management. It allows resource provisioning from a centralized location, and the service provider is able to control the network through a centralized interface.

- Dynamic control: finally, since SDN is software-based, it allows the users to control both the physical-level resources and the virtual-level resources through the control plane. Dynamic network paths configuration and proactively network services de-

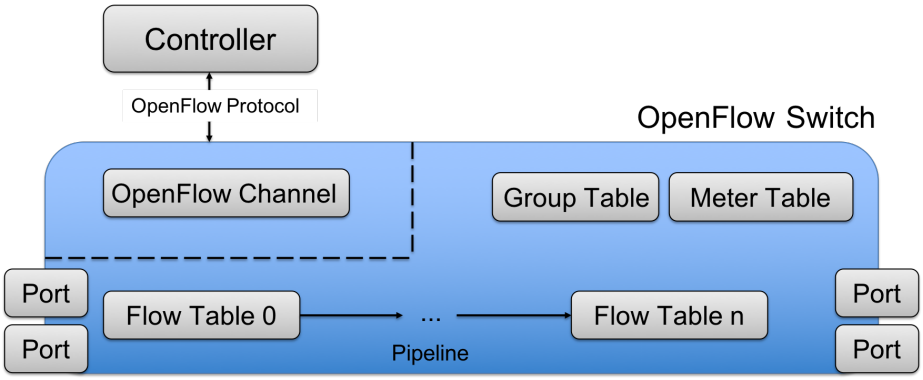


Figure 2.4: OpenFlow switch architecture

ployment become possible.

The different aspects of the benefits brought by SDN incorporate together with each other and provide a new picture of future networking. In summary, with SDN, the introduction of network innovations can be very fast and the management of large networks can be radically simplified and automated.

2.2.2 OpenFlow Protocol

Managed by the ONF, OpenFlow [26] is the *de facto* southbound protocol defining the communication routine between the control layer and infrastructure layer and is usually viewed as the enabler of SDN. It defines the switch components, such as ports, flow table and group table, the switch functionalities, such as packet matching and pipeline processing, and the interactions between switch and controller, such as statistics collection and feature collection. The OpenFlow Switch Specification has been published from version 1.0 to version 1.5 in the year 2011-2015.

As shown in Fig. 2.4, an OpenFlow switch integrates one or more flow tables, which perform packet lookups and forwarding, an OpenFlow channel to an external controller, and the OpenFlow protocol. The flow entries containing in a flow table are composed of the match fields, counters, and a set of instructions. A packet is matched against the match fields, which include packet headers, the ingress port, and the metadata value. The

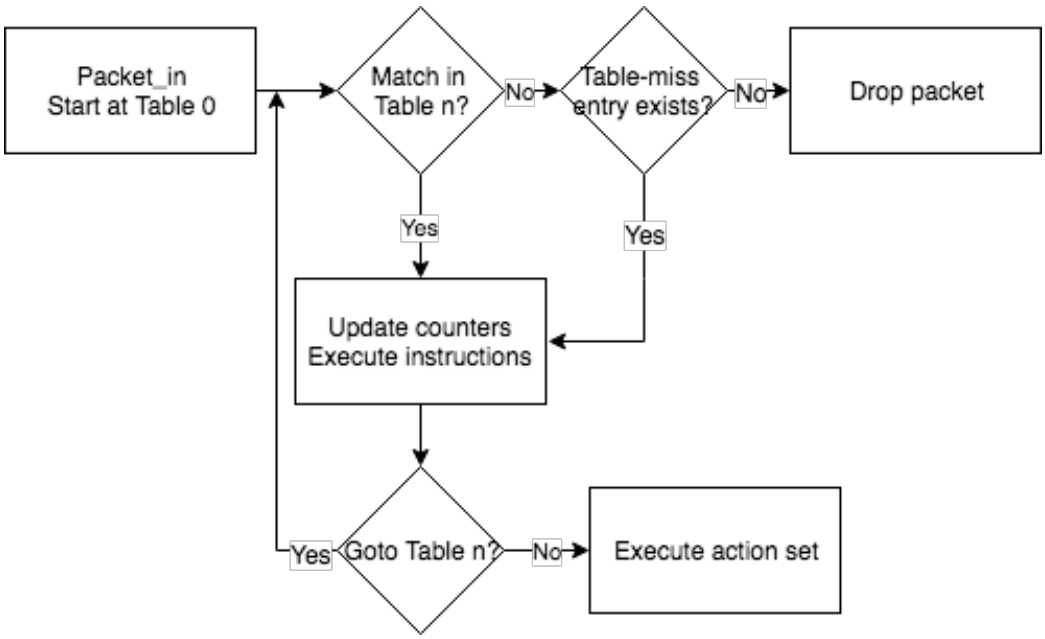


Figure 2.5: Packet matching process

number of times a particular entry being matched is recorded using the counter field, which is essential for the statistics collection function. Possible instructions for a matched flow include sending the packet out through a certain port, modifying some fields in the packet, simply dropping the packet, etc. The secure channel remotely connects the OpenFlow switch to the controller. The communication between switch and controller is standardized by the OpenFlow protocol.

As one of the core functionalities of OpenFlow switches, the packet matching process is shown in Fig. 2.5. When an OpenFlow-enabled switch receives a packet from data plane, it iterates through its flow tables to look for matched flow entry. If a matched entry exists, the switch will execute corresponding instructions and update the counter. If the switch cannot find a matched entry and the table-miss flow entry is set as sending to controller, which is the most common configuration for the table-miss entry, the switch will send a *packet_in* message to the controller via the switch-to-controller link. Otherwise, the packet will get dropped. After the controller receives the *packet_in* message, it will set up corresponding routing path, install flow entries on involved switches, and instruct the switch to forward the

matched packets according to the entry. If the flow table is full and the new flow does not match with any existing entries, the OpenFlow switch will evict some existing flow entries to install new flow entries. However, when the packets from the evicted entries come into the network, the switch will identify them as unmatched flows and send *packet_in* message to the controller again. This switch-controller communication will cause extra delay for the packet. Besides packet matching and routing path setup, the controller can also perform functionalities like statistics collection and state maintenance via the OpenFlow protocol to manage the network.

However, before this new paradigm can be widely deployed in enterprises, the scalability issues caused by the limited resources at SDN switches should be addressed.

2.3 Resource Constraints of Software-defined DCN

As discussed in previous chapter, to take full advantage of the benefits brought by SDN and achieve optimized network resource allocation, the new resource constraints coming with the SDN architecture need to be examined and coped with in the resource allocation algorithms for software-defined DCN.

2.3.1 Switch-to-Controller Link

In order to address the switch-to-controller link capacity limitation, proactive entry installation is proposed to reduce the switch-to-controller load [27]. However, proactive flow entry installation goes against the key benefits of SDN, such as flexible network management. Besides, the switch-to-controller link overflow problem is still unsolved when a large number of unmatched flows come into the network or when frequent statistic collections are needed. Meanwhile, researchers also try to optimize the control applications to reduce the interactions between switches and controller. For example, a new routing scheme is proposed in [28] to reduce the number of network events need to be sent to the controller. The results show that this new scheme can reduce the network events handled in the control

plane by 430% compared to a traditional routing scheme in SDN. In [29], the flow's path information is encapsulated into the packets' header at the source switches. This prevents the interaction between the controller and the intermediate switches along the routing path during path setup. Moreover, some researchers also propose to utilize multiple controllers in SDN to provide a scalable and resilient network operation, and the optimized controller placement algorithms are studied to reduce the switch-to-controller latency [30, 31, 32, 33]. Yet utilizing multiple controllers will induce extra overhead and higher maintenance cost. For example, the multiple-controller cases generate 30%-50% messaging overhead due to synchronization and controller-to-controller communication in [30]. Corresponding strategies are required to mitigate the side effects with multiple controllers in SDN. Last but not least, auxiliary connections were introduced from OpenFlow 1.3 [26] to exploit the parallelism of switch implementation and alleviate the switch-to-controller link overflow. Some academic works propose that the auxiliary connections are also beneficial for failure restoration and path migration [34, 35, 36]. The auxiliary connection is already enabled in many open-source SDN controllers such as FloodLight [37] and OpenDaylight [38], but its performance is not fully studied yet with the DCN implementation. In [39], the OpenFlow 1.3 model of network simulator ns-3 [40] is extended to support multiple TCP/UDP auxiliary connections. However, a single simulation run with realistic data center traffic loads would take days of computation time. In this context, an analytical model which produces evaluation results for realistic DCN scale within a reasonable time is indispensable.

2.3.2 Flow Table

The flow tables of OpenFlow switches are usually placed on expensive and power-hungry TCAM, which achieves single-clock-cycle lookup time. It is obvious that the flow table capacity is limited due to power, cost, and chip size constraints. Yet there is another hidden cost of highly consumed flow tables. M. Kuniar et al. [41] discover that the performance of OpenFlow switches drops with an increasing number of installed rules even before the

flow table is full. Therefore, maintaining the flow table size at a reasonable level is of significant importance in terms of both avoiding flow table overflow and retaining normal switch performance.

In order to address this issue, intelligent and adaptive time-out setting mechanisms for flow entries are proposed. A suitable setting on the time-out value can avoid unnecessary *packet_in* events caused by deleting the entry prematurely, and at the same time reduce the number of invalid flow entries in the flow table. The combination of a heuristic algorithm to compute adaptive time-outs and a mechanism to proactively evict the flow rules is proposed in [42], with the objective of effective utilization and low misses of TCAM. Besides, assigning proper time-outs to flows according to their traffic characteristics in DCN [43], and determining the lower and upper bound of time-out values for Instant Messaging [44] are also proposed. Finally, flow aggregation is proposed in some works to reduce the number of required flow entries in the network. For example, since the traffic engineering algorithms cannot operate at the granularity of individual applications, Google's WAN B4 [45] aggregates applications to flow groups defined by *(src, dst, application QoS)* for scalability. According to the results in [46], with flow rule compression considering the QoS of traffic, the number of required entries reduces by around 30% for the simulated topology.

2.4 Resource Allocation Optimization in SDN

Different methods and techniques have been studied to improve network resource allocation using SDN. In order to improve resource allocation in the cloud environment, a bandwidth allocation approach that satisfies QoS requirements for all priority cloud users based on SDN is proposed in [47]. Application-aware resource allocation scheme is also studied in [48] to predict resource requirements and allocate an appropriate number of virtual machines for each application in SDN-based cloud data center. A. Ghosh et al. [49] examine two traffic management designs to optimize sending rates at the flow-level across multiple traffic classes in software-defined data center backbone networks. As for the SDN-based

wireless network, S. Namal et al. [50] study the load balancing algorithm leveraged by flow admission control with SDN in the 5G network and achieve more than 200% of per-flow resource allocation. Moreover, an SDN-based resource allocation framework is proposed in [51] to properly orchestrate heterogeneous radio bandwidth in emerging LTE/WLAN multi-radio networks in a centralized and holistic manner. The aforementioned research works are focused on leveraging the benefits brought by SDN to optimize network resource allocation, but they do not consider the new resource constraints brought by the SDN architecture.

Traditional route selection algorithms usually focus on optimizing the objectives with the link capacity constraint. However, the link utilization level on all the links keeps low in data center [6]. On the other hand, the flow table size is limited as previously discussed. Thus the route selection algorithms proposed for SDN should take the flow table capacity constraint into consideration. In order to achieve better resource allocation for SDN with the new resource constraints, some researchers propose to optimize the routing rules to reduce and balance the flow table usage. In [52], R. Cohen et al. concentrate on satisfying global network objectives, such as maximum flow, in environments where the size of the forwarding table in network devices is limited. Dynamic routing algorithms to maximize network throughput, by jointly considering the flow table capacity at each switch and the bandwidth capacity at each link, for both unicast and multicast cases are studied in [53]. The controller processing capacity constraint is added to the SDN throughput maximization problem for the first time in [54].

Besides seeking to allocate a limited set of resources to a set of individuals with demands and targeting at maximizing a global network objective, fairness is another common objective in resource allocation optimizations. J. Zhang et al. in [55] propose to optimize with max-min fairness in SDN multi-path routing with the joint consideration of the forwarding rule placement. A price-based joint allocation model and a fair allocation algorithm of link bandwidth and flow table for multiple control applications in SDN are

presented in [56]. The proportional fair allocation of the link bandwidth and the minimum global delay are obtained at the same time. The Multi-Class QoS-Guaranteed Traffic Management (MCTEQ) algorithm presented in [57] is a constrained utility-optimization formulation of the joint-bandwidth allocation problem for multiple classes of traffic in inter-data center communication. The flows allocation of the same class satisfies proportional fairness and the classes with higher priority such as the interactive traffic have a larger weight in the optimization problem. Finally, the SWAN algorithm presented in [48] also allocates bandwidth in a max-min fair manner within a traffic class.

With a lot of ongoing research to tackle the network resource allocation problem for SDN, there are still unexplored areas in this direction. First, an intelligent admission control mechanism determining how to admit incoming flows into the network with congested flow tables is still required. M. Huang et al. in [53] propose to admit a new flow only when there is a routing path to meet its resource demands. In [58], Z. Guo et al. also propose to reject the new flow if every pre-generated path contains a switch with a fully-loaded flow table, arguing that this action can improve network throughput and reduce the controller's workload. However, simply dropping the new flows when there is not enough resource will degrade the user experience and cause profit loss of the DCN service provider. The admission/rejection decision should be made to maximize the overall benefits for all users, including both the users of the existing flows and the users of the new flows. The trade-off between the throughput of existing flows and the acceptance rate of new flows should be considered. Therefore, a more systematic admission control mechanism in a software-defined DCN taking the flow table capacity into consideration is in need. Besides, as one of the resource constraints of software-defined DCN, the limited bandwidth capacity of the switch-to-controller link is not considered in the previously proposed route selection algorithms. High flow inter-arrival rate to a switch will cause intensive message exchanges between switch and controller. As stated in previous section, with auxiliary connections implemented, different switches will have various switch-to-controller link

capacities. This should be considered as one of the constraints when designing routing rules for the network. Finally, in order to achieve application-requirement-aware resource allocation, the expression of the application requirements, such as end-to-end delay, in SDN environment need to be reconsidered due to the unique network architecture.

2.5 Chapter Summary

In this chapter, we first look into the increasing usage of cloud computing and DCN, and the new network capabilities desired. Then we compare the traditional network architecture and the SDN architecture. We also review the *de facto* southbound protocol, OpenFlow of SDN and its packet matching process. Finally, we examine the new resource limitations brought by the novel SDN architecture and the resource allocation optimization algorithms in SDN.

CHAPTER 3

QUEUEING ANALYSIS OF AUXILIARY-CONNECTION-ENABLED SWITCHES

3.1 Introduction

The large gap between the data center demand and the current switch-to-controller link bandwidth will cause packets loss and prolonged flow setup delay, and degrade the network performance. In order to address this challenge, auxiliary connections were introduced from OpenFlow 1.3 [26] to exploit the parallelism of switch implementation. By default, the channel between an OpenFlow switch and controller is a single network connection. OpenFlow 1.3 proposed to create auxiliary connections besides the main connection between switches and the controller. According to the specification, the controller is free to use the various switch connections for sending OpenFlow messages at its entire discretion. The auxiliary connection is already enabled by many open-source SDN controllers such as OpenDaylight. Besides, some academic works also propose auxiliary connections are beneficial. However, the auxiliary connection is not fully studied yet at the switch side. Therefore, an understanding on the potential of auxiliary connections in improving network performance is a prerequisite for its real-world data center deployment. In this context, an analytical model which produces evaluation results with realistic data center traffic loads is indispensable.

Analytical models have been developed for SDN [59, 60, 61, 62, 63, 64], but they are limited in the following aspects. First, none of them considers auxiliary connections in their model. Second, the only interaction considered in these works is the *packet_in* process. They omit the *feature/stats_request/reply* interactions. Furthermore, only [62] considers a realistic OpenFlow switch with limited buffer space. Without a realistic assumption of the OpenFlow switch, the models will not be able to capture the accurate network performance,

such as packet loss rate. Finally, the validation results are not provided in [59]. In this context, we propose our model to compensate for these limitations.

In this chapter, we develop a queueing model for auxiliary-connection-enabled switches with limited buffer space. Since the OpenFlow Specification has been very vague about the actual implementation, to test its performance, we assume that the auxiliary connections behave the same as the main connection and they are all referred to as multiple connections in this chapter. When a new packet from the data plane comes to the switch, if the switch cannot find a matched entry, it sends a *packet_in* message to the controller via the switch-to-controller link. It can either only send the packet header with the payload buffered at the switch, or send the whole packet to the controller without buffering it depending on the switch implementation. The first approach suffers from the limited switch buffer size, while the second one suffers from the limited bandwidth of the switch-to-controller link. With auxiliary connections implemented and the switch-to-controller link limitation addressed, the second approach is advantageous since it avoids switch buffer overflow. Therefore in this work, we focus on the second approach, which is also utilized by OpenvSwitch 2.7 and after [65]. The model captures not only the *packet_in* interaction, but also the *stats/feature_request/reply* interactions, which are of great importance to enable SDN benefits, such as efficient resource allocation. Equations are derived to evaluate the network performance. The model is validated for lower traffic loads using the network simulator ns-3 [10], where network protocols including OpenFlow are implemented. Our model is then used to perform an in-depth analysis of switch performance under realistic data center traffic loads. In particular, we thoroughly characterize the packet loss rate, average number of packets in switch, and flow setup delay in the network. With limited hardware resource available at a switch to build auxiliary connections, our model can serve as a tool to predict network performance and provision switches with the appropriate number of connections to satisfy the network QoS requirements. To the best of our knowledge, this is the first analytical model of auxiliary-connection-enabled OpenFlow switches.

Table 3.1: Notations for the queueing model of the auxiliary-connection-enabled switches

$S0$	Queue node at switch to collect arriving packets
$S1$	Queue node at switch to send packets to controller
C	Queue node at controller
W	Number of connections between switch and controller
λ_S	Packet arrival rate to $S0$ from data plane
λ_C	Packet arrival rate of the controller-generated packets
μ_{S0}	Service rate of $S0$
$\mu_1, \mu_2, \cdots, \mu_W$	Individual service rates of the servers in $S1$
μ_{S1}	Overall service rate at $S1$
μ_C	Service rate of C
β	Probability that the packets not sent to controller
α	Probability that the packets are finished at controller
N	Switch buffer size
i, j, k	The number of packets at queue node $C, S0$ and $S1$
$\mathbf{T}$	Multi-dimensional state space
$\mathbf{Q}$	Transition rate matrix
π	Steady-state probability vector
TH_C, TH_{S0}, TH_{S1}	Average departure rate at $C, S0$ and $S1$
L_C, L_{S0}, L_{S1}	Average number of customers in $C, S0$ and $S1$
L_S	Average number of packets in switch
P_S	Packet loss rate of the system
T_{setup}	Flow setup delay
T_{prop}	Two-way propagation delay

3.2 Analytical Model

In this section we develop and validate the queueing model of auxiliary-connection-enabled switches as shown in Fig.3.1 and the parameters are listed in Table 3.1.

3.2.1 Queueing Model

The switch is modeled as a two-node queueing network composing of $S0$ and $S1$. Node $S0$ collects all the incoming messages to the switch while $S1$ sends messages to the controller. External packets from the network arrive at $S0$ according to a Poisson process with rate λ_S. Its service time is exponentially distributed with rate μ_{S0}. We assume that the probability of packets not being sent to controller is β, e.g., packets from the matched flows. The messages need to be sent to the controller, e.g., packets from the unmatched flows, will be

sent to $S1$. The multiple connections between the switch and controller are modeled as W servers of $S1$, with exponentially distributed service times at rates $\mu_1, \mu_2 \cdots \mu_W$. The total buffer space at the switch is limited at N and packet loss will happen if the buffer space is full.

The controller is modeled as a single queue node C with service time, following exponential distribution at rate μ_C, which includes both the controller processing time and the transmission time to switch, and is positively correlated with W since the transmission time is dependent on the number of connections. The incoming traffic to C is made up of the messages from the switch, (such as *packet_in*, *feature/stats_reply*) and the controller-generated messages (such as *flow_mod, feature/stats_request*). The second type of messages is generated according to a Poisson process with rate λ_C. The incoming traffic to C either causes a corresponding message to be sent to switch, (e.g., a *packet_out* message will be sent to the switch for each *packet_in* message) or will be finished at the controller (e.g., the controller collects the statistics with the *stats_reply* messages). We assume the probability that the incoming messages are finished at controller is α. Parameter λ_C and α varies based on the number of *feature/stats_request/reply* messages in the network, and is used to model these interactions. Since SDN controller is implemented on a high-power server, we assume the buffer space of C is unlimited. This assumption will be justified in the evaluation section by the approximated average number of total packets at the controller.

3.2.2 Model Analysis

This queueing model can be analyzed as a three-dimensional continuous time Markov chain (CTMC) using quasi-birth-death process [66]. Its multidimensional state space $\mathbf{T}$ is given as set of tuples:

$$\mathbf{T} = \{(i, j, k) | i, j, k \in \mathbb{N}_0, j + k \leq N\},$$

where i, j, k is the number of packets in node $C, S0$, and $S1$. With i as the level variable, the transition rate matrix of the CTMC can be grouped into finite sub-matrices with block

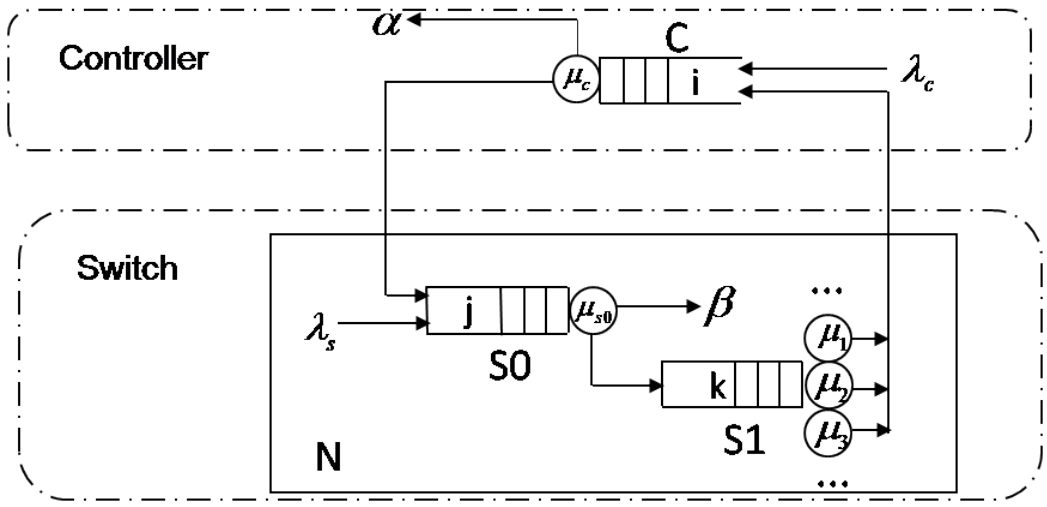


Figure 3.1: Queueing model of auxiliary-connection-enabled switches for SDN. The switch is modeled as a two-node queueing network composing of $S0$ and $S1$. The controller is modeled as a single queue node C.

structures: B_0, A_0, A_1, and A_2, which is given as:

$$\mathbf{Q} = \begin{array}{c} \\ T_0 \\ T_1 \\ T_2 \\ T_3 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} T_0 & T_1 & T_2 & T_3 & \cdots \end{array} \\ \begin{pmatrix} B_0 & A_0 & \mathbf{0} & \mathbf{0} & \cdots \\ A_2 & A_1 & A_0 & \mathbf{0} & \cdots \\ \mathbf{0} & A_2 & A_1 & A_0 & \cdots \\ \mathbf{0} & \mathbf{0} & A_2 & A_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array},$$

where T_i is a subset of $\mathbf{T}$ with the same i value. Moreover, B_0 is the transition rate matrix within level $i = 0$, and A_2, A_1, A_0 are transition rate matrices from level i to level $i - 1$, within level i, and from level i to level $i + 1$, respectively. Taking j as the second level variable, each of the matrices can be further grouped into sub-matrices to determine its elements. The detailed steps are shown in the following.

A_0

If the value of i increases by 1, the incoming packet at C can come from the controller-generated messages or from the switch, and thus the value of k does not change or decreases by 1, respectively. The value of j does not change in both cases. Therefore, A_0 can be structured as:

$$A_0 = \begin{array}{c} \\ T_{i,0} \\ T_{i,1} \\ T_{i,2} \\ \vdots \\ T_{i,N} \end{array} \begin{array}{c} \begin{array}{ccccc} T_{i+1,0} & T_{i+1,1} & T_{i+1,2} & \cdots & T_{i+1,N} \end{array} \\ \begin{pmatrix} C^0 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & C^1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & C^2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & C^N \end{pmatrix} \end{array}.$$

Matrix C^j is dependent on the second level variable j and is a square matrix with size $(N+1-j) \times (N+1-j)$.

$$C^j_{(k,k') \in \mathbb{N}_{N-j} \times \mathbb{N}_{N-j}} = \begin{cases} \lambda_C, & \text{if } k^{'} = k \\ \mu_{S1}, & \text{if } k^{'} = k-1 \\ 0, & \text{Otherwise} \end{cases},$$

where $\mathbb{N}_n$ denotes the set of natural numbers $\{0, 1, \cdots n\}$, and μ_{S1} is the overall service rate at queue node $S1$:

$$\mu_{S1} = \begin{cases} \mu_1 + \cdots + \mu_k, & \text{if } k < W \\ \mu_1 + \cdots + \mu_W, & \text{Otherwise} \end{cases}.$$

A_2

If the value of i decreases by 1, the outgoing message either goes to $S0$ or is finished at C, and thus the value of j increases by 1 or does not change, respectively. The value of k does not change in both cases. Thus A_2 is:

$$A_2 = \begin{array}{c} \\ T_{i,0} \\ T_{i,1} \\ T_{i,2} \\ \vdots \\ T_{i,N} \end{array} \begin{array}{c} \begin{array}{ccccc} T_{i-1,0} & T_{i-1,1} & T_{i-1,2} & \cdots & T_{i-1,N} \end{array} \\ \begin{pmatrix} D^0 & E^0 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D^1 & E^1 & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & D^2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & D^N \end{pmatrix} \end{array},$$

where

$$D^j_{(k,k')\in\mathbb{N}_{N-j}\times\mathbb{N}_{N-j}} = \begin{cases} \alpha\mu_C, & \text{if } k' = k \\ 0, & \text{Otherwise} \end{cases},$$

and

$$E^j_{(k,k')\in\mathbb{N}_{N-j}\times\mathbb{N}_{N-j-1}} = \begin{cases} (1-\alpha)\mu_C, & \text{if } k' = k \\ 0, & \text{Otherwise} \end{cases}.$$

D^j is a square matrix with size $(N+1-j)\times(N+1-j)$ and matrix E^j has size $(N+1-j)\times(N-j)$. E^j is not a square matrix to make sure that the sum of j and k is still within the switch buffer limitation after the state change.

A_1

If the value of i does not change, the value of j can increase by 1, decrease by 1, or do not change. Thus, A_1 is structured as following:

$$A_1 = \begin{array}{c} \\ T_{i,0} \\ T_{i,1} \\ T_{i,2} \\ \vdots \\ T_{i,N} \end{array} \begin{array}{c} \begin{array}{ccccc} T_{i,0} & T_{i,1} & T_{i,2} & \cdots & T_{i,N} \end{array} \\ \begin{pmatrix} G^0 & H^0 & \mathbf{0} & \cdots & \mathbf{0} \\ F^1 & G^1 & H^1 & \cdots & \mathbf{0} \\ \mathbf{0} & F^2 & G^2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & G^N \end{pmatrix} \end{array}.$$

If i does not change and j decreases by 1, the output message of $S0$ can either be forwarded to $S1$ with probability $1-\beta$, or be finished at switch with probability β. Thus

$$F^j_{(k,k')\in\mathbb{N}_{N-j}\times\mathbb{N}_{N-j+1}} = \begin{cases} \beta\mu_{S0}, & \text{if } k' = k \\ (1-\beta)\mu_{S0}, & \text{if } k' = k+1 \\ 0, & \text{Otherwise} \end{cases}.$$

Next, we derive the $(N+1-j)\times(N-j)$ matrix H^j, where j increases by 1 and k does not change. The incoming packet to $S0$ is from the external network since i does not change.

$$H^j_{(k,k')\in\mathbb{N}_{N-j}\times\mathbb{N}_{N-j-1}} = \begin{cases} \lambda_S, & \text{if } k' = k \\ 0, & \text{Otherwise} \end{cases}.$$

The $(N+1-j)\times(N+1-j)$ matrix $G^j_{(k,k')\in\mathbb{N}_{N-j}\times\mathbb{N}_{N-j}}$ is a diagonal matrix, whose elements stand for the transition rates where the values of i, j, k do not change. It ensures

that all elements in each row of $\mathbf{Q}$ sum up to 0. Its diagonal elements are:

$$G^j_{(k,k)} = \begin{cases} -\lambda_C - \lambda_S - \mu_C, & \text{if } j = 0 \text{ and } k = 0 \\ -\lambda_C - \lambda_S - \mu_C - \mu_{S1}, & \text{if } j = 0 \text{ and } 0 < k < N \\ -\lambda_C - \lambda_S - \mu_C - \mu_{S0}, & \text{if } k = 0 \text{ and } 0 < j < N \\ -\lambda_C - \mu_{S1} - \alpha\mu_C, & \text{if } j = 0 \text{ and } k = N \\ -\lambda_C - \mu_{S0} - \alpha\mu_C, & \text{if } k = 0 \text{ and } j = N \\ -\lambda_C - \mu_{S1} - \alpha\mu_C - \mu_{S0}, & \text{if } 0 < j < N \text{ and } k = N - j \\ -\lambda_C - \lambda_S - \mu_C - \mu_{S1} - \mu_{S0}, & \text{if } 0 < j < N \text{ and } 0 < k < N - j \end{cases}$$

B_0

Matrix B_0 has the same structure, and sub-matrices $F^{0,j}, H^{0,j}$ as F^j, H^j in A_1. The only difference is the diagonal matrix $G^{0,j}$, where

$$G^{0,j}_{(k,k)} = \begin{cases} -\lambda_C - \lambda_S, & \text{if } j = 0 \text{ and } k = 0 \\ -\lambda_C - \lambda_S - \mu_{S1}, & \text{if } j = 0 \text{ and } 0 < k < N \\ -\lambda_C - \lambda_S - \mu_{S0}, & \text{if } k = 0 \text{ and } 0 < j < N \\ -\lambda_C - \mu_{S1}, & \text{if } j = 0 \text{ and } k = N \\ -\lambda_C - \mu_{S0}, & \text{if } k = 0 \text{ and } j = N \\ -\lambda_C - \mu_{S1} - \mu_{S0}, & \text{if } 0 < j < N \text{ and } k = N - j \\ -\lambda_C - \lambda_S - \mu_{S1} - \mu_{S0}, & \text{if } 0 < j < N \text{ and } 0 < k < N - j \end{cases}$$

This CTMC process is stable if $\pi_A A_0 \mathbf{1} < \pi_A A_2 \mathbf{1}$, where π_A is the steady-state probability vector of the generator matrix $A = A_0 + A_1 + A_2$, and $\mathbf{1}$ is a column vector of 1's. With a stable CTMC and the transition rate matrix $\mathbf{Q}$ constructed, the steady-state

probability vector π can be calculated. First, we partition π according to the level variable i:

$$\pi = [\vec{\pi}_0, \vec{\pi}_1, \cdots]^T,$$

where

$$\vec{\pi}_i = \big[\pi_{i,0,0}, \pi_{i,0,1}, \cdots, \pi_{i,0,N}, \pi_{i,1,0}, \cdots, \pi_{i,1,N-1}, \cdots, \pi_{i,N,0}\big].$$

The global balance equations for $i > 0$ is:

$$\vec{\pi}_{i-1}A_0 + \vec{\pi}_i A_1 + \vec{\pi}_{i+1}A_2 = 0. \tag{3.1}$$

Since $\vec{\pi}_i$ can be defined in terms of $\vec{\pi}_{i-1}$ and the transitions between the levels are independent of the level number i, a constant rate matrix R is defined to lead to the matrix-geometric equation:

$$\vec{\pi}_i = \vec{\pi}_{i-1}R = \vec{\pi}_1 R^{i-1}, i > 0. \tag{3.2}$$

Substitute Equation (3.2) into Equation (3.1), we obtain

$$A_0 + RA_1 + R^2A_2 = 0 \tag{3.3}$$

to solve R. To calculate R, we use the Logarithmic Reduction algorithm described in [67]. Finally, the steady-state probability vector $\vec{\pi}_i$ can be obtained using R, the boundary condition:

$$\vec{\pi}_0 B_0 + \vec{\pi}_1 A_2 = 0,$$
$$\vec{\pi}_0 A_0 + \vec{\pi}_1 A_1 + \vec{\pi}_2 A_2 = 0,$$

and the fact that the sum of all elements in π is 1.

3.2.3 Performance Analysis

With the steady-state probability calculated, we derive the equations for main network performance metrics, including the packet loss rate (P_S), average number of packets in switch (L_S), and the flow setup delay (T_{setup}).

The average departure rates at $C, S0$, and $S1$ are:

$$TH_C = \mu_C(1 - \sum_{(0,j,k)\in T} \pi_{0,j,k}),$$
$$TH_{S0} = \mu_{S0}(1 - \sum_{(i,0,k)\in T} \pi_{i,0,k}),$$
$$TH_{S1} = \mu_{S1}(1 - \sum_{(i,j,0)\in T} \pi_{i,j,0}).$$

In stationary state, the incoming and outgoing traffic at switch obeys the following relation:

$$[\lambda_S + (1-\alpha)TH_C](1 - P_S) = \beta * TH_{S0} + TH_{S1},$$

and thus the loss rate at switch can be derived. The average number of customers in each queue node (L_C, L_{S0}, L_{S1}) and in the switch (L_S) is:

$$L_C = \sum_{(i,j,k)\in T} i \times \pi_{i,j,k},$$
$$L_{S0} = \sum_{(i,j,k)\in T} j \times \pi_{i,j,k},$$
$$L_{S1} = \sum_{(i,j,k)\in T} k \times \pi_{i,j,k},$$
$$L_S = L_{S0} + L_{S1}.$$

The flow setup delay is defined as the time interval between the time that the first packet of a new flow arrives at the switch and the time it is forwarded by the switch to network. It

Table 3.2: Parameter settings for validation and performance evaluation

Parameters	Validation	Evaluation
λ_C [packets/s]	70	$10k$
μ_C [packets/s]	$220W$	$120000W$
λ_S [packets/s]	$25 - 370$	$100k - 1000k$
μ_{S0} [packets/s]	2000	$1000k$
$\mu_1, \mu_2, \cdots$ [packets/s]	$100, 100, \cdots$	$50k, 50k, \cdots$
W [connections]	$[1, 2, 3]$	$[1, 2, 3]$
N [packets]	20	$10 - 120$
α	1%	10%
β	80%	$87\% - 99\%$

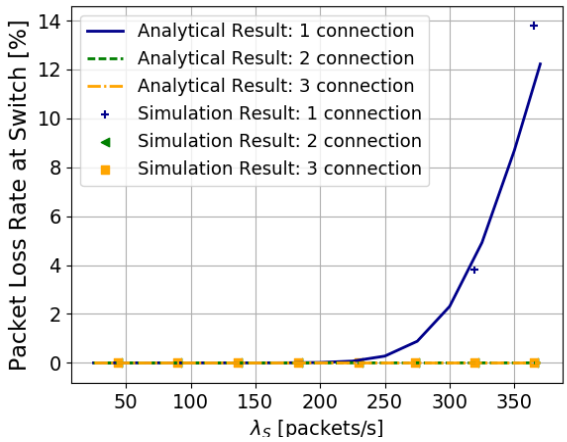
is composed of the queueing delays at the switch and controller and the propagation delay. Based on Little's Law [68],

$$T_{\text{setup}} = \frac{L_{S0}}{TH_{S0}} + \frac{L_{S1}}{TH_{S1}} + \frac{L_C}{TH_C} + \frac{L_{S0}}{TH_{S0}} + T_{prop},$$

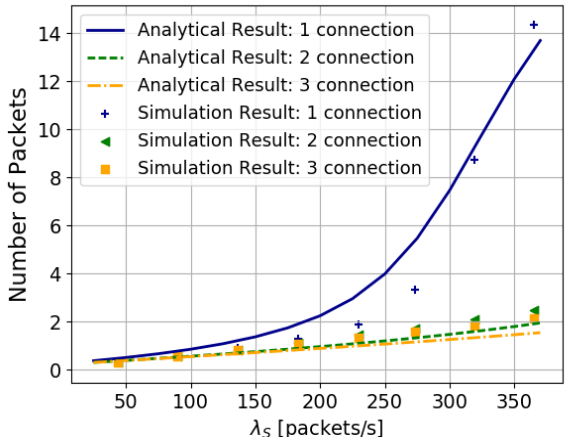
where T_{prop} is the two-way propagation delay.

3.2.4 Multiple-Switch Network

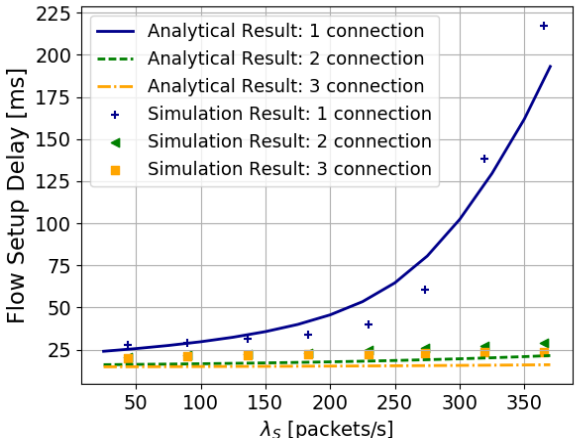
To utilize the switch model in network with M switches, we assume that for a certain message sent from controller to switch, the probability that the destination is switch $1, 2, \cdots, M$ is $p_1, p_2, \cdots, p_M$ and $\sum_{m=1}^{M} p_m = 1$. Thus, the transition rate where switch m receives a packet from controller is $p_m(1-\alpha)\mu_C$, and transition rate where C receives a packet from switches is $\sum_{\text{m=1}}^{M} \mu_{S1}^m$. The network becomes a $2M+1$ dimensional CTMC and the performance of each switch can be analyzed following similar procedure as presented above. In this chapter, the analysis and equation derivation is detailed for the single-switch case for readers to easier understand the model.



(a) Packet loss rate



(b) Number of packets at switch



(c) Flow setup delay

Figure 3.2: Validation for analytical model

3.3 Validation through Simulation

We first confirm the mathematical solution using Python simulation [69] and the parameters are in the "Validation" column of Table 3.2. The simulated queueing model reflects the structure shown in Fig.3.1. Our analytical results for P_S, L_S and T_{setup} matched exactly with the simulation results validating the correctness of our analyses.

In order to validate our queueing model in a realistic network environment, we used ns-3 to simulate an SDN network with OpenFlow and auxiliary connections implemented [39]. There is one OpenFlow switch with varying number of connections to the controller. There are 200 hosts attached to the switch and each host starts to send traffic randomly. In the simulation, we use a realistic traffic generator on each host to relax the assumption of the Poisson arrival process used in the analytical model. The traffic is ON/OFF and the bit rate is constant. The actual packet inter-arrival rate to switch (λ_S) is calculated based on the simulation results. The "Validation" column of Table 3.2 lists some additional parameters. The results are shown in Fig.3.2, where each data point is obtained based on ten simulation runs using different random seeds. With real-world incoming traffic pattern, the simulation results match well with the theoretical ones indicating that our analytical model is able to capture the trend of performance changes. The slight mismatch may come from the difference in the traffic pattern. Besides, the specific implementation of utilized network protocols, such as TCP congestion control, will also impact the actual performance.

In view of the fact that a single simulation run for a realistic-scaled data center network takes hours to finish and consumes a great amount of memory, in both parts of the validation, we scale down the number of events in the simulation to obtain the validation results in an appropriate amount of time. This fact strengthens the need for an analytical model to obtain a comprehensive analysis.

3.4 Network Performance Evaluation

In this section, we use our analytical model to evaluate switch performance with 1–3 switch-to-controller connections. The parameter settings are from the "Evaluation" column in Table 3.2. Note that the packet inter-arrival rate is more than 3 orders of magnitude larger than the value used in this section. With this larger inter-arrival rate, Python simulation would take several days, the ns-3 simulation would need almost one week, but evaluation using our analytical model required only seconds. We first approximate the average number of packets at controller to justify our assumption of adequate buffer space in controller. Then we vary λ_S, $1-\beta$ (the probability that an incoming packet goes to the controller), and N to evaluate the network performance. Evaluations such as these can be used to provision switches with an appropriate number of connections based on the expected traffic characteristics within a deployed network.

3.4.1 Number of Packets in Controller

In order to justify our assumption on the unlimited buffer space of controller, we measure the number of packets in controller with varying packet arrival rates to the switch as shown in Fig. 3.3. First, with more number of connections between switch and controller, the packets will arrive at the controller from switch at a higher rate. However, at the same time, the more number of connections also enable faster transmission from controller to switch, so with the same packet arrival rate to the switch, the average number of packets at the controller decreases with the increasing number of switch-to-controller connections. For the one-connection case, when the packet arrival rate is greater than $1000k$ packets/s, the system becomes unstable due to the limited processing capability at controller. In this case, the number of packets queueing at the controller will become infinity and cannot be modeled. When the packet arrival rate is $1000k$ packets/s and the system is stable, the number of packets at the switch for the one-connection case is around 72. Since each switch

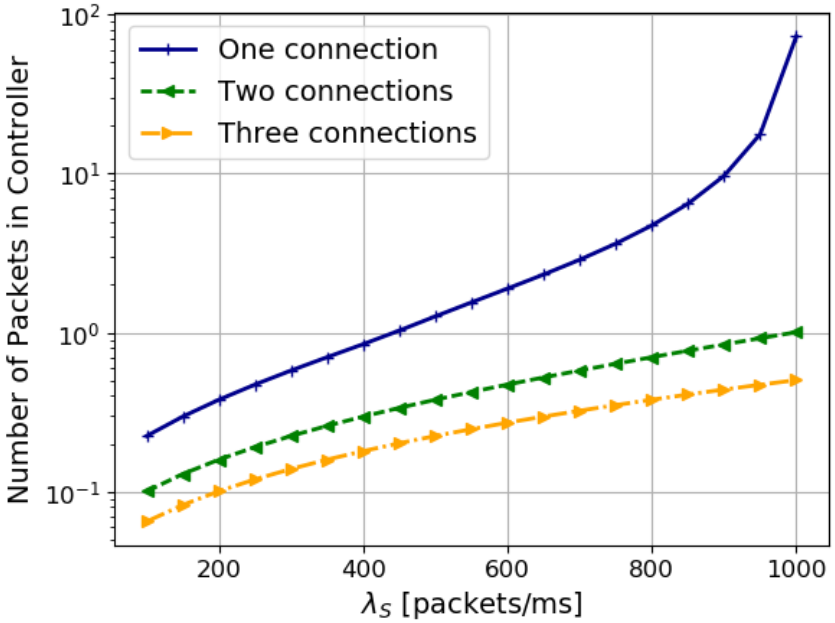


Figure 3.3: Average number of packets at controller

will have separate connections to the controller, for a DCN with 100 switches, the total number of packets at the controller in this worst case can be estimated as $72 \times 100 = 7200$. Assuming that each packet is 100 Byte, the required memory space is around 0.7 MB, which can be achieved in any servers. The packet arrival rate at $1000k$ per switch is a high enough number to represent most data centers. Therefore, it is reasonable to assume that the buffer space at the controller is adequate even in this worst case scenario.

3.4.2 Varying Packet Arrival Rate

As shown in Fig.3.4a, the packet loss rate of the single-connection case increases fast when $\lambda_S > 400k$ packets/s and reaches around 50% when $\lambda_S = 900k$ packets/s. On the other hand, the loss rate does not increase until λ_S reaches $850k$ packets/s for the multiple-connection cases. The three-connection case outperforms the two-connection case but the improvement (around 1%) is minimal. In this case, the capacity limitation exists in $S0$, so increasing the number of connections in $S1$ cannot improve the performance significantly.

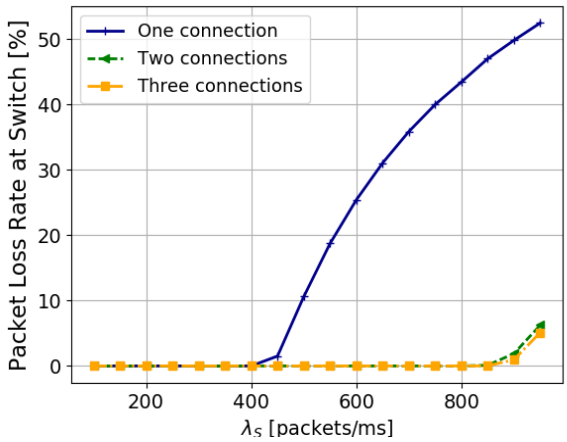
Overall, with different traffic load, different number of connections are recommended.

We use the buffer utilization level to indicate the average number of packets stored in switch (shown in Fig.3.4b). With a smaller packet arrival rate ($\lambda_S < 400k$ packets/s), the switch buffer utilization level stays at a relatively low level for all cases, which corresponds with a zero packet loss rate. The switch buffer becomes more utilized as λ_S increases, and finally is bounded by the buffer size limitation. A high utilization level indicates that the switch is full most of the time and thus high loss rate is caused. When the utilization level is greater than 10%, all the cases start to induce packet loss.

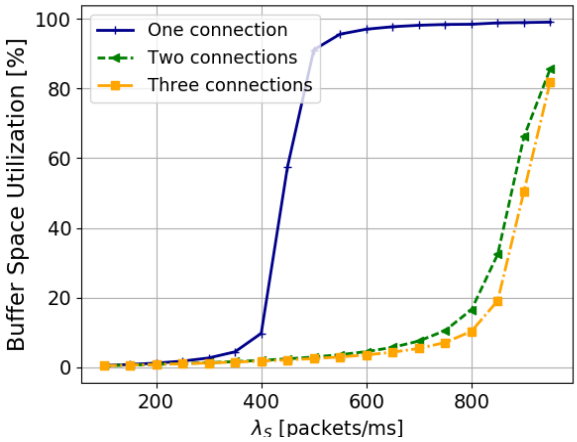
Fig.3.4c shows the average flow setup delay. Multiple connections can improve the flow setup delay by 37-50% with varying λ_S. According to the queueing theory, the queueing delay depends on the buffer size. Thus when the buffer utilization level is greater than 90%, the flow setup delay of the single-connection case stays at a fixed value.

3.4.3 Varying $1 - \beta$

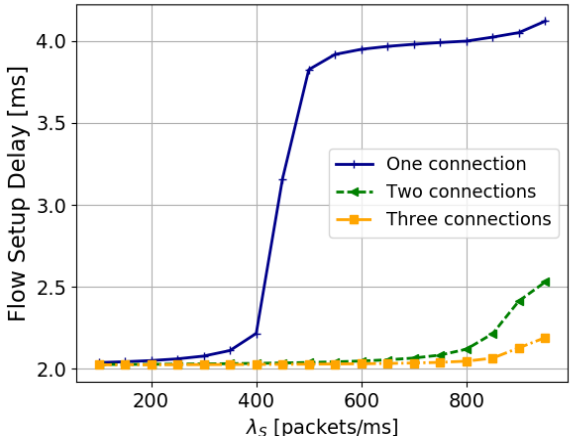
The value of $1 - \beta$ depends on the network traffic characteristics. For example, if the average flow length in the network is low, the probability that a packet is the first packet of a flow and needs to go to the controller is high, which means $1-\beta$ is high. In general, when the probability of the incoming packets going to the controller increases, the performance (as shown in Fig.3.5) persists a similar trend as λ_S increases since they both burden $S1$. However, while the value of λ_S impacts the overall load on the switch, including both $S0$ and $S1$, the change of $1 - \beta$ will induce different load to $S1$ specifically. Thus, a higher processing rate of $S1$ can improve the performance significantly. The performance improvement induced by utilizing more connection is much more obvious compared with the case when we vary λ_S. For example, when $1 - \beta = 12\%$, the three-connection case induces a notably better performance for all the metrics than the two-connection case.



(a) Packet loss rate

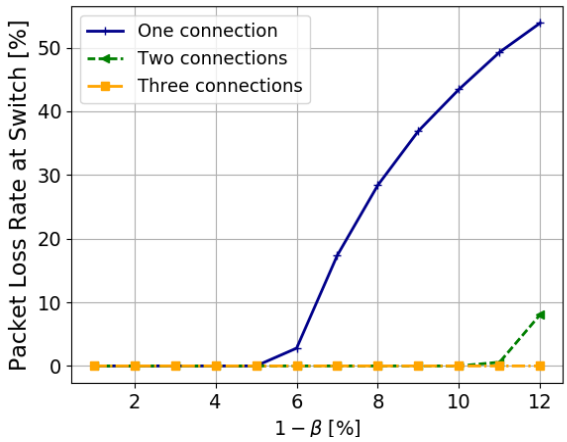


(b) Switch buffer utilization

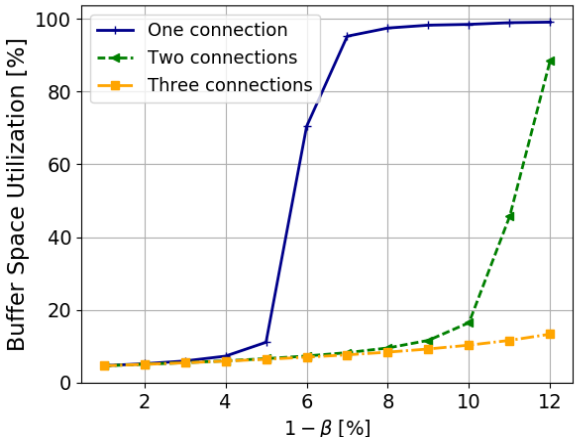


(c) Flow setup delay

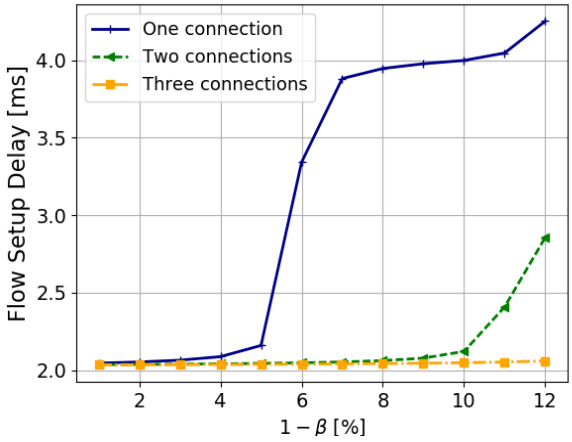
Figure 3.4: Network performance with varying λ_S, $N = 100$, $\beta = 90\%$



(a) Packet loss rate

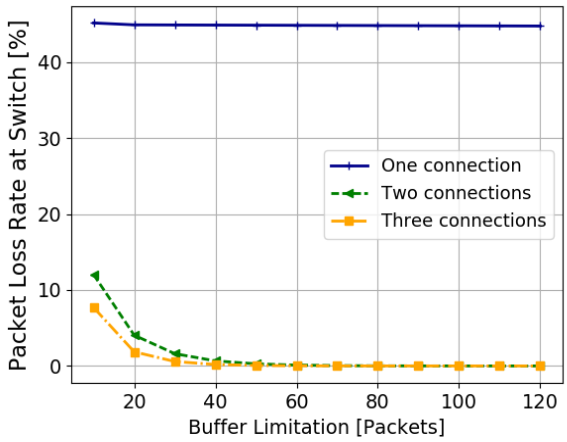


(b) Switch buffer utilization

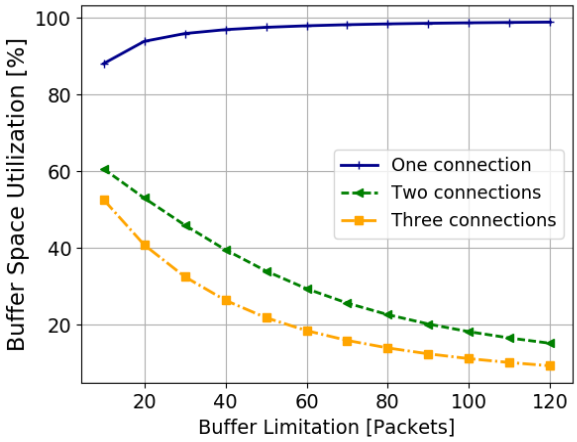


(c) Flow setup delay

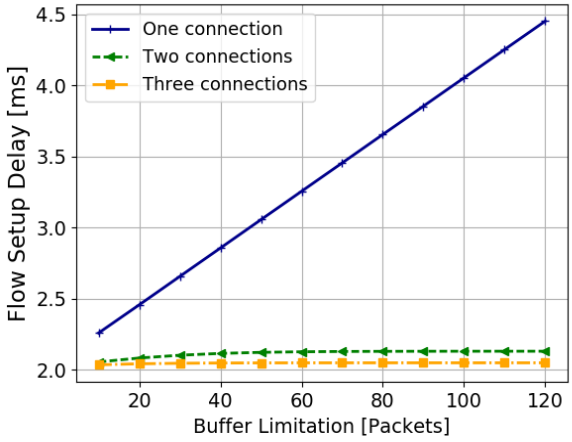
Figure 3.5: Network performance with varying $1 - \beta$, $\lambda_S = 800k$, $N = 100$



(a) Packet loss rate



(b) Switch buffer utilization



(c) Flow setup delay

Figure 3.6: Network performance with varying N, $\lambda_S = 800k$, $\beta = 90\%$

3.4.4 Varying Switch Buffer Space

As shown in Fig.3.6a, as the switch buffer size increases, the multiple-connection cases can decrease the loss rate by 30-45% compared with the single-connection case and achieve a stable service. With varying buffer space available in the switch, the loss rate with single-connection case stays at 50%, indicating more buffer space is required to provide a stable service. In general, with more connections in use, the switch requires less buffer to provide a stable service.

In terms of buffer utilization level as shown in Fig.3.6b, the switch with only one connection tends to be fully occupied all the time, and thus the packet loss rate stays high. Cases with more connections always generate a lower utilization level. For example, the three-connection case incurs around 10% less utilization than the two-connection case.

Fig.3.6c shows that the flow setup delay of the single-connection case increases linearly with the switch buffer size. This linear increase can be justified by queueing theory, which states that a larger buffer size allows more packets waiting in the queue and thus may cause a longer delay. With only one connection, more buffer space does not help with the packet loss rate, but generates a longer flow setup delay instead. On the other hand, with multiple connections, the switch maintains a low buffer utilization level, and thus the flow setup delay stays at a lower level.

In summary, the single-connection case requires much larger switch buffer size to maintain a stable service than the multiple-connection cases. However, larger buffer size might incur longer flow setup delay. Furthermore, with different traffic characteristics, different number of connections are recommended to improve the network performance. Our model can serve as a tool to find the minimum number of required connections.

3.5 Chapter Summary

In this chapter, we propose and validate an analytical queueing model for auxiliary-connection-enabled OpenFlow switches. This is the first known work to analytically study auxiliary connections in SDN. An in-depth analysis on the network performance with varying number of connections between OpenFlow switch and controller is also carried out. Our analysis can be used in the network deployment phase to provision switches with the appropriate number of auxiliary connections based on expected traffic characteristics. We believe this model lays the foundation of the deployment of OpenFlow auxiliary connections.

CHAPTER 4

ADMISSION CONTROL IN VIEW OF FLOW TABLE CAPACITY

4.1 Introduction

The large gap between the flow table capacity and the data center demand causes prolonged end-to-end delay of packets and possible congestion on the switch to controller link. With the default flow table management mechanism defined in OpenFlow 1.3 Switch Specification, if a switch cannot find any space in the requested table to add the incoming flow entry, the switch will send a message to controller and drop the new entry. At the same time, the incoming packet can be forwarded based on the *packet_out* message. This naive approach ensures that the existing flow entries will not get dropped, but this will induce continuous *packet_in* messages to the controller since every packet from the new flows cannot be matched by the full switches. It burdens the controller with unnecessary processing and causes resource waste at switch if some existing flows are inactive. Although the flow entries of the existing flows are not removed, the computation and storage resources consumed by the new flows will affect the existing flows.

Admission control is a process to check if the current resource is sufficient to accept the new request. In the traditional network, admission control algorithms have been studied for years. On-line and off-line algorithms were proposed in [70, 71] to gain the maximum profits and balance the link bandwidth in the network. With the benefits brought by the centralized controller, several research works focused on using SDN to control the flow admission in the network. In [72], J. Huang et al. proposed a model for admission control with flow aggregation. S. Namal et al. in [50] utilized two admission control mechanisms, namely ”cut-off priority” and ”fractional guard channel” in SDN. J. Leguay et al. in [73] adapted some key admission control algorithms from the literature in SDN and proposed an

admission control meta-algorithm using machine learning technique. However, the aforementioned works focus on proposing an algorithm to make a decision on whether to accept the incoming flow in view of the link utilization and none of these works consider the flow table capacity limitation in SDN. SDN flow table overloading attacks and mitigation mechanisms were studied in [74] and [75]. They proposed to leverage the unused flow table space in other switches to install new rules when a switch runs out of its own flow table. R. Cohen et al. in [52] focused on maximizing the network utilization where the network forwarding table is limited, but the algorithm can only handle the static scene. A K Similar Greedy Tree (KSGT) algorithm was proposed in [76] to maximize the number of flows in the network, constrained by the limited flow table space in SDN switches. However, an admission control mechanism determining when to admit incoming flows into the network if the flow table is congested is still required when all the active flows are legitimate.

In this chapter, we propose and analyze a lightweight admission control mechanism for SDN. It controls the flow entrance to the network when the flow table is congested. The admission control mechanism is implemented at the controller level and strives to provide the best QoS to most users and generate good benefits for service providers. Simulations are carried out using a canonical three-tier data center topology and the mechanism is thoroughly evaluated in terms of data plane and control plane performance.

4.2 Flow Table Overloading in SDN

The flow table overloading problem in software-defined DCN is analyzed in this section. We elaborate on the flow table capacity in SDN and some simple flow table management mechanisms.

4.2.1 Flow Table Capacity

According to the statistics collected in [6], the number of active flows at an access switch of a datacenter ranges from 1000 to 5000 about 90% of the time. If the flow entry timeout is 60

seconds, an average access switch might have roughly 78000 flow entries [1]. Besides, the aggregation switches and core switches will install more flow rules than the edge switches. This large demand for flow table capacity incurs scalability problems in data center. First, when a switch with full flow tables receives a *flow_mod* message from the controller, it needs to decide if an existing flow entry should be removed to accommodate the new entry. This results in the contention between new and existing flows. The QoS of the flows cannot be guaranteed if their entries are removed from the flow table. The extra queuing and processing delay induced by its removed entry may degrade the users' experience. Second, the *packet_in* message will be generated again when future packets from the removed flow come in. Repeated *packet_in* message processing consumes the computation and storage resources at both controllers and switches, and limits their capabilities to provide service to other users. In addition, the repeated *packet_in* messages will cause congestion at the southbound interface and lead to packet loss. The transmission queue from the switch to the controller has finite length and will drop the packets if it is full [1]. Almost all the traffic in the data center utilizes TCP protocol, and thus the packet loss at switch will result in limited data rate of the traffic and extra congestion in data plane to transmit lost packets.

4.2.2 Existing Flow Table Management

The default flow table management mechanism defined in the OpenFlow Specification is described in the previous section. It will cause performance degradation and resource waste. A simple improvement is to remove the least recently used (LRU) entry in the flow table to accommodate the incoming entries if a switch is full. In general, the LRU entry is highly possible to be inactive. However, this is not guaranteed to be true. If all the flow entries are active, removing LRU entry will behave similarly as dropping the new entry. In addition, iterating through the flow tables to look for the LRU entry incurs extra processing. Moreover, implementing this removing-LRU-entry table management goes against the founding principle of SDN to delegate all the intelligence to the controller. Thus, a

Algorithm 1 Admission Control Algorithm

Initialize: $switches \leftarrow$ [A, B, C, ...]
Initialize: $uncongested\_switches \leftarrow$ [A, B, C, ...]
Initialize: $rejected\_count \leftarrow [\,]$
Initialize: $max \leftarrow max\_no\_retries$
procedure FLOW TABLE UTILIZATION MONITOR
    for all $switch \in switches[\,]$ **do**
        if flow table utilization of $switch >$ threshold **then**
            remove $switch$ from $uncongested\_switches[\,]$
        else
            add $switch$ to $uncongested\_switches[\,]$
        end if
    end for
end procedure
procedure PACKET_IN EVENT
    $route, out\_port \leftarrow calculate\_route(flow, uncongested\_switches[])$
    if $route = NULL$ **then**
        $rejected\_count[flow] + 1$
        if $rejected\_count[flow] > max$ **then**
            $route, out\_port \leftarrow calculate\_route(flow, switches[])$
            $rejected\_count[flow] = 0$
        end if
    end if
    if $route = NULL$ **then**
        $packet\_out \leftarrow Drop$
    else
        $packet\_out \leftarrow out\_port, flow\_mod \leftarrow$ *route*
    end if
end procedure

more intelligent admission control mechanism implemented in the controller is required to improve the SDN scalability and the QoS of the network traffic.

4.3 Admission Control in Software-defined DCN

The admission control mechanism implemented in data center should be lightweight considering the large amount of active flows. Besides, balancing the flow table usage while admitting flow entries to a switch is always more efficient compared to regulating the existing entries. Finally, the controller should admit a flow only when the switches on the

route which the flow will go through can offer enough flow table space. Otherwise, if the flow is rejected at a bottleneck switch, the flow table space at other switches is unnecessarily consumed. Based on these principles, we propose a controller level admission control mechanism in this section. It focuses on restricting new flow entrance on congested switches while maintaining the data plane performance.

The algorithm is introduced in Algorithm 1. The controller will keep track of the congested switches and the number of times being rejected for each flow ($rejected\_count$). First, The flow table utilization level is monitored by the controller using the *flow_removed* messages or the flow table statistics from switches. Many controller functions already require such information from switches so little overhead will be induced by our proposal. When the flow table utilization level at a switch exceeds a threshold, we define the switch as congested. Otherwise, the switch is uncongested. Flow table capacity at congested switches is reserved by setting the threshold. Second, when receiving a *packet_in* message, the controller will calculate a route with only the uncongested switches for the flow. This ensures that the flow entries are balanced among the switches. It also makes sure once the flow is admitted, it will not be dropped by other bottleneck switches, and thus avoids further congestion at intermediate switches along the route. In brief, routes do not pass the congested switches will be enforced in the network. Furthermore, if all the switches are congested and no alternate route can be found ($route = NULL$), the controller will instruct the switch to simply drop the packets and no new entry will be installed. Note that the dropped packets are usually the initial packets of the flows (e.g., ARP Request, SYN message for TCP flows). In this way, the servers will try to initialize the flow again after timeout. If the former congested switches become available, the flow will be admitted to the network and the corresponding flow entries will get installed. 50% of the flows in the data center last less than 0.1 seconds [6], so the possibility that the congested switches become available is fairly high. At the same time, existing flows will not be affected. This avoids the repeated *packet_in* messages if the flow is admitted at its first try. If the server has

attempted for max number of retries and the switches are still congested, the controller will admit the flow using the reserved capacity to avoid further delay. If there are multiple controllers in the network, the switch information can be synchronized among the controllers during the collaboration process, and thus the algorithm is feasible in this case. Depending on the multi-controller architecture, the admission control mechanism can be deployed accordingly. In the following sections, we particularly focus on the single controller scenario. In the following sections, we elaborate on the results of simulation experiments conducted in a data center network and compare them with respect to data plane and control plane performance.

4.4 Simulation Setup

Experimentations have been performed using the network simulation software ns-3 to evaluate the performance of the proposed admission control mechanism.

4.4.1 Data center Topology

The network topology used in the simulation, as shown in Fig. 4.1, is a canonical three-tier data center topology. Each of the access layer switches is connected to 15 servers. A single node acts as the core switch and represents the gateway to other data centers and the Internet. In this work, the number of aggregation switches and access switches is set to be 3 and 16. The total number of servers in the topology is 240, which can represent a common scale of a university data center [6]. Network links are all wired links with the following characteristics:

- Server to Access Switch: 100 Mbps, 1 ms delay
- Access to Aggregation Switch: 10 Gbps, 1 ms delay
- Aggregation to Core Switch: 10 Gbps, 1 ms delay
- Switch to Controller: 17 Mbps [1], 1 ms delay

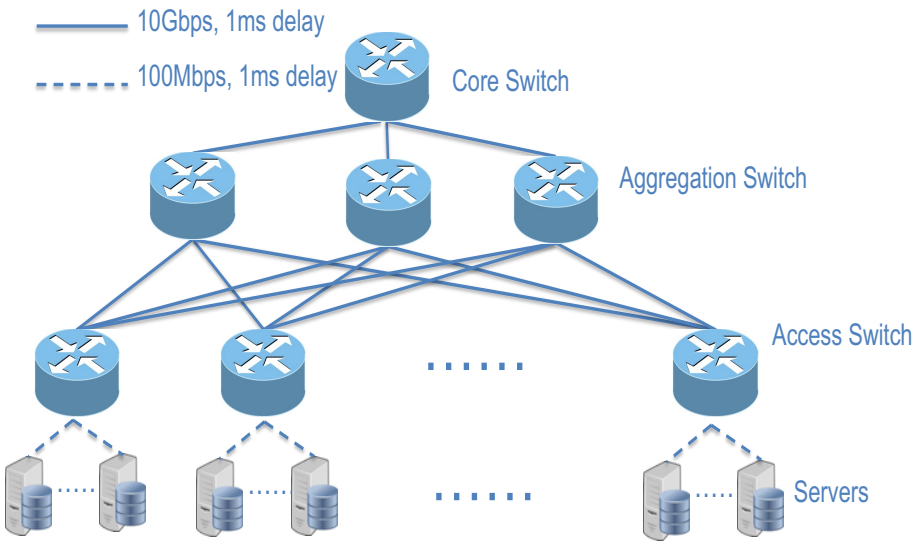


Figure 4.1: Data center network topology

The data center traffic in the simulation is set according to real-world measurements in [6]. The transport layer protocol of traffic in data centers is composed mostly by TCP and 80% of the flow inter-arrival times are between 4ms and 40ms. In our simulation environment, each flow is targeting at either another server or the Internet. The average flow inter-arrival time is set to be 4 ms, 8 ms, 16 ms, 32 ms with a Poisson distribution. The application data rate is 500 kbps.

4.4.2 SDN Implementation

The OpenFlow switch is implemented using OFSWITCH13 module [40]. It includes the switch network device, the controller application interface, the OpenFlow channel, and the external *ofsoftswitch13* library. The flow table management is implemented using the *ofsoftswitch13* library. The flow table capacity is defined as 1000, which is a typical value according to [1]. Flow entries will be removed after the flows end to guarantee that all the entries in the switch are active. All the switches are controlled by a self-implemented controller running the Dijkstra's algorithm to calculate the routes in the topology. The controller is connected to the switch through the controller application interface provided

by OFSWITCH13. The threshold is set to be 90% and the max number of retries is 3 based on simulation test results. We are particularly interested in the robustness of our proposed mechanism in terms of different traffic inter-arrival time in the network. Different inter-arrival time indicates different traffic load in a unit time.

4.5 Results and Discussions

In this section, we analyze the simulation results to evaluate the performance of the proposed admission control mechanism. We compare the data plane and control plane performance between three cases: with default flow table management, with admission control mechanism, and the unlimited case where the flow table capacity is sufficient. The unlimited case serves as a reference in the evaluation but it is not feasible in the real-world implementation.

4.5.1 Data Plane Performance

To evaluate the data plane performance, we measure the total received data, the TCP congestion window and goodput, the TCP connection attempts before the service is established, and the number of services provided to the users.

Total received data

The service provider of data centers will get revenue based on the amount of data transmitted in the network. As shown in Fig. 4.2, we capture the total received data over a 300 seconds interval for the three cases. As expected, the unlimited case achieves the highest amount of received data. Our admission control mechanism receives around 95% of the unlimited case while with default management, the received data is only around 75%. The admission control mechanism increases the total received data by 23-25% compared to the default management. The mechanism performs stably regarding different flow inter-arrival time and can handle high flow arrival rate.

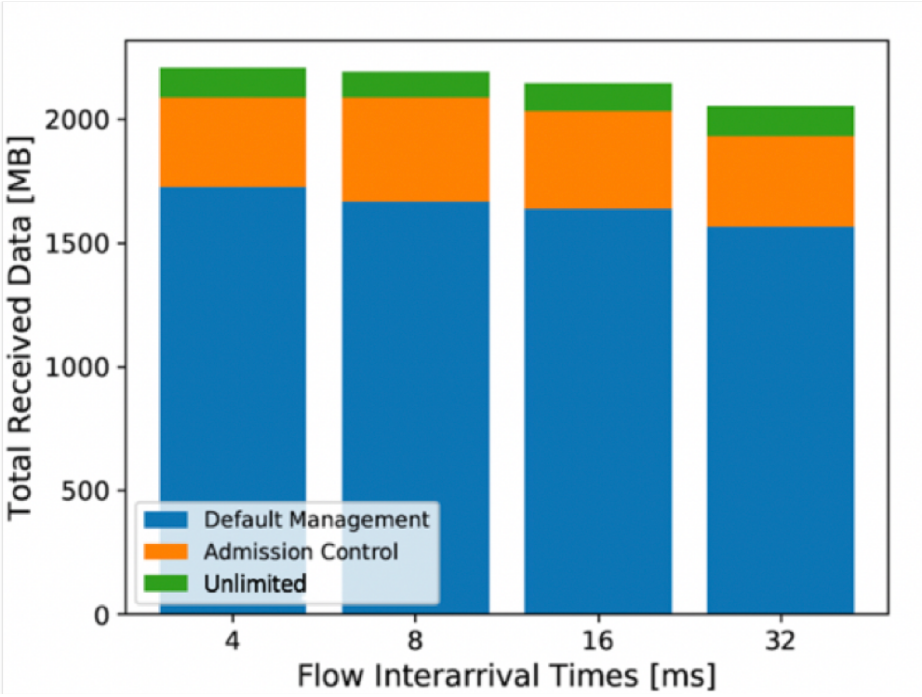
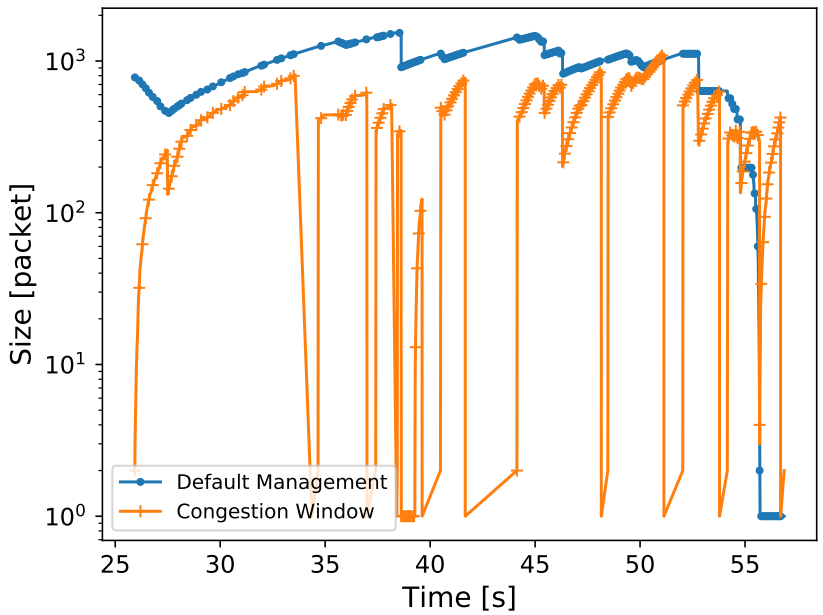


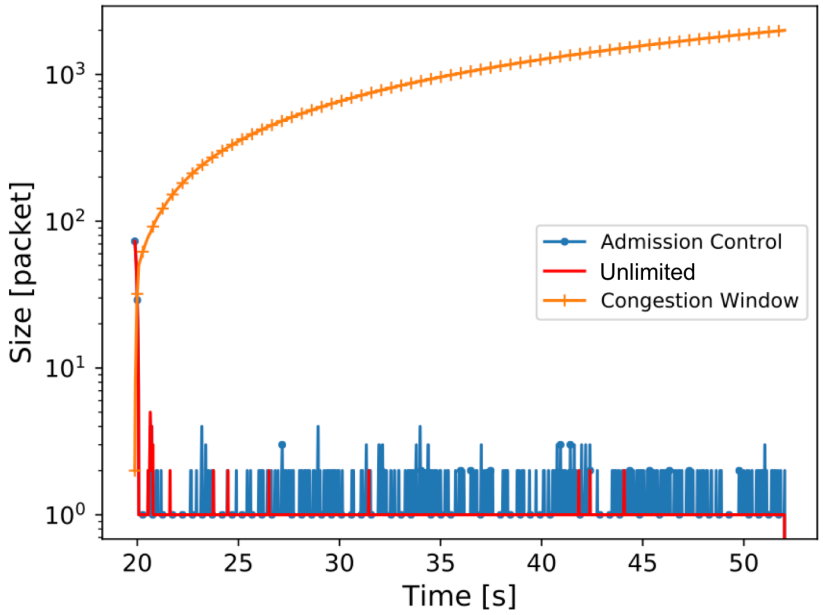
Figure 4.2: Total received data

TCP congestion control and goodput

One of the service quality provided to the data center clients is goodput. Goodput is defined as the amount of useful data delivered in a unit time. We study the goodput performance and its relation to the TCP congestion control mechanism. Originally, the TCP congestion avoidance mechanism is designed to avoid congestion in the data plane. It assumes network congestion if a packet is lost. However, with SDN-based network, the packet loss at switch indicates congestion at the southbound interface and will also lead to a shrinking congestion window of the sender. We monitor the congestion window of the same flow when the inter-arrival time is 32ms and plot it in Fig. 4.3. The data waiting to be sent at the TCP socket is also displayed. The difference between the congestion window and waiting data for each case indicates the data plane performance. With the default table management, the flow is set up successfully at 25.2 seconds, which is 6 seconds after the flow is initiated. The congestion window starts to grow but packets loss happens throughout the flow transmission process and the seesaw oscillations are observed. The waiting data accumulates at



(a) Packet loss rate



(b) Switch buffer utilization

Figure 4.3: The congestion window and buffered data waiting to be sent (a) with default flow table management, and (b) with admission control and unlimited case.

the TCP socket since packets loss happen and congestion window size varies. In this case, the throughput is limited by the congestion window size. Notice that the goodput will be lower than throughput since the retransmitted packets are duplicates of the lost ones. With the unlimited and admission control cases, the congestion window keeps increasing, indicating no packet loss at both data plane and southbound interface. This suggests that the packets loss in the default management case happens at the southbound interface since all three cases employ the same data plane capacity. The waiting data is smaller than the congestion window size so the goodput is primarily limited by the application data rate. This analysis can be verified by their throughput and goodput. The throughput of the default management case is 1130 kbps while its goodput is only 435 kbps. The throughput and goodput for the other two cases are both 499 kbps, which is 1kbps less than the application data rate. The results indicate that our mechanism can reduce packet loss at the SDN switch and improve goodput.

Request success rate and TCP connection attempts per request

In order to illustrate the user experience of clients, we also measure how many services are provided to the users successfully and how many TCP connection attempts are initiated before the service is established. There are 2880 service requests throughout the simulation. The average request success rate of the unlimited case, admission control case, and default management case is 99.8%, 98.1%, and 56.0% respectively. Our proposal increases the number of provided service by 42% compared to the default management case, and performs as good as the unlimited case. Regarding to the number of TCP connection attempts before the connection is established, the measurements are collected for the successfully established requests and showed in Table 4.1. More TCP attempts indicate a longer service setup delay. The number for the unlimited case persists around the same value for different flow inter-arrival time, which indicates that the congestion level at the data plane stays the same. As the flow inter-arrival time decreases, the average number of TCP connection

Table 4.1: Average number of TCP connection attempts

Flow Inter-arrival Time	4 ms	8 ms	16 ms	32 ms
Unlimited	1.5	1.4	1.3	1.3
Admission Control	2.6	2.3	1.8	1.4
Default Management	3.3	2.8	2.7	2.6

attempts for the other two cases increases since the SDN southbound interface is more congested due to the limited capacity of the flow table. The admission control mechanism can reduce an average of one TCP attempt per request from the default management case.

4.5.2 Control Plane Performance

To evaluate the control plane performance, we measure the total number of *packet_in* events to the controller and the average controller processing delay.

Number of packet_in events

Table 4.2 shows the number of *packet_in* events processed by the SDN controller. The number of the default management case is less than the other two cases since some *packet_in* messages are dropped at the switch transmission queue and around 40% of the service is not established as shown in Section 4.5.1. The number for the unlimited case stays around 19160 for varying flow inter-arrival time. Our admission control algorithm incurs 4% to 18% more *packet_in* events than the unlimited case. This is because some flows are not admitted to the network when the flow table is congested initially. Their retries for connection will generate extra *packet_in* events. As the flow inter-arrival time increases, the overhead on the number of *packet_in* events decreases.

Average controller processing delay

The controller is running on a workstation with a 4-core 3.2GHz Intel i5-6500 processor and 7.7 GiB memory. We define the controller processing delay as the time interval between the controller receives the *packet_in* message from switches and sends out the

Table 4.2: Number of packet_in events

Flow Inter-arrival Time	4 ms	8 ms	16 ms	32 ms
Unlimited	19168	19165	19166	19160
Admission Control	22727	20463	20232	20018
Default Management	12611	12360	13639	15522

Table 4.3: Controller processing delay [ms]

Flow Inter-arrival Time	4 ms	8 ms	16 ms	32 ms
Unlimited	3.4	1.8	2.5	2.8
Admission Control	8.6	5.3	4.9	3.4
Default Management	0.3	0.4	0.5	0.4

corresponding *packet_out* and *flow_mod* messages. The measurements are summarized in Table 4.3. The default management case has the lowest processing delay due to its relatively small number of *packet_in* events. The prolonged processing delay of the admission control case is incurred by the higher number of *packet_in* events and extra processing delay to calculate routes for flows at controller. However, based on the measurements from [77], the controller response time, which includes the traversal of networking stacks on both the controller and Cbench sides twice, as well as the processing time of the controller, can reach to hundreds of milliseconds. The controller processing overhead caused by the admission control is insignificant in this case.

4.6 Chapter Summary

In this chapter, we present a lightweight controller level admission control mechanism for software-defined DCN in view of the flow table capacity at switches. It utilizes only the flow information and statistics collected by the existing controller functionalities. Moreover, an in-depth performance evaluation was carried out based on a canonical data center topology. In particular, we found that the current flow table management fails to provide service to 44% of the users and has a relatively long service establishment delay. With a tolerable number of *packet_in* events and controller processing delay, the proposed mechanism increases the number of services provided by 42% and improves the network goodput

by reducing the number of packet loss at switches. The total number of data received is also increased by 23-25%. We believe this work lays out the foundation for admission control in a software-defined DCN.

CHAPTER 5
HIGH SATISFACTION AND FAIR ALLOCATION OF RESOURCES

5.1 Introduction

In view of the two new resources constraints brought by the SDN architecture, new resource allocation algorithms in software-defined DCN need to be investigated. In general, high demand satisfaction and fairness are two fundamental objectives in resource allocation that cannot be maximized simultaneously. This motivates the investigation of inherent trade-offs between the two objectives, where a common approach is to maximize demand satisfaction subject to some fairness constraints. The existing literature in the field of SDN resource allocation is discussed as follows. Some research works propose to maximize the network flows or network throughput in SDN with limited flow table size [52, 54], without considering any fairness constraints among flows. A heuristic algorithm is designed in [55] to maximize the minimum throughput satisfaction. That work considers only a simple max-min fairness model and does not allow trade-offs between fairness and satisfaction. The aforementioned algorithms only consider the flow table size constraint while the control channel bandwidth is another important constraint for software-defined DCNs. The controller processing capacity constraint is added to the SDN throughput maximization problem in [54], but the control channel bandwidth limitation is still omitted. Moreover, these works focus on maximizing network throughput with limited flow table space. However, in current DCNs, the data plane link utilization level is relatively low [6], and thus the throughput demands of users can be easily satisfied. On the other hand, the resource constraints brought by the unique architecture of SDN are likely to become the bottleneck. Optimization problems with the objective to improve SDN resource usage have not been well explored.

In this chapter, we address the resource allocation problem in software-defined DCNs, with the objective to maximize the total satisfaction ratio of the aforementioned two SDN resources, subject to different fairness constraints. Our approach is to aggregate individual flows into flow groups and then find the optimal routing paths and the corresponding resource allocation vectors for each flow group satisfying our objective. Our main contributions are as follows:

- We address the resource allocation problem coupled with satisfaction maximization, fairness constraints, and routing path selection in software-defined DCNs. The networks can better meet resource demands when a combination of fair and efficient allocation algorithms and intelligent routing decisions is employed. Therefore, instead of fixing the routing and allocating the resources subject to fairness constraints, we include routing path selection into the optimization problem to better meet the resource demands. The evaluation results show that deviating from simple shortest-path routing can improve the satisfaction ratio by 25%-30% while maintaining very good fairness.

- We investigate different fairness models, including the classical max-min fairness, simple max-min fairness, and equal share fairness model. The fairness models are also modified to cooperate with our two-resource satisfaction maximization case. Moreover, to accommodate various network requirements, we introduce a relaxation parameter $\delta \in [0, 1]$ into these fairness models. It allows the network operator to control the trade-off between total demand satisfaction and fairness.

- We consider joint optimization of the two SDN resources, which leads to better network utilization compared with single-resource optimization.

5.2 System Model

In this section, we introduce our system model. The parameters used are listed in Table 5.1. We consider a scenario where the software-defined DCN is modeled as a graph $G = (V, E)$ and the flows from each server are aggregated into flow groups at the source switches based on their destination switches. The set of all aggregated flow groups $F = [f_1, f_2, \cdots, f_N]$ and f_n is defined as:

$$f_n = (s_n, t_n, D_b(n), D_{fe}(n), D_{sc}(n)),$$

where s_n and t_n is the source and destination switch of f_n, and $D_b(n), D_{fe}(n)$, and $D_{sc}(n)$ are the total data plane bandwidth demand, flow table demand, and control channel bandwidth demand requested by f_n, respectively. The previous traffic statistics or the real-time measurement at switch. can be used to form the demands of flow groups. The flow aggregation step will not only reduce the computation complexity of the optimization problem significantly, but also help reduce the flow table usage. While SDN enables individual flow management, this fine-grained control will cause flow table overflow problem. Therefore, coarse-grained control is used in some cases. For example, in the software-defined WAN deployed by Google, the flows are aggregated to groups defined by $\{src, dst, QoS\}$ for scalability [45]. In this work, we assume that with the flow aggregation, some flow rules (e.g., rules for the flows with the same source and destination address, and QoS class, or some mice flows that do not need full control) can be compressed together. With OpenFlow-enabled switches, the QoS requirement can be achieved by the OpenFlow queue and meters. The number of flow rules demanded after aggregation depends on the number of QoS classes enforced in the network, and level of control granularity the service providers prefer. The flow entries of the same flow group can be placed on the same flow table to enable easier management for switch. The parameter $D_{sc}(n)$ is the control channel bandwidth required to set up the flow entries, collect the statistics, and maintain the flows. Although these two demands in network are time-varying and the value of $D_{sc}(n)$ is af-

Table 5.1: Notations for system model in Chapter 5

v	Switch in SDN
e	Data plane link in SDN
V	All switches in SDN
E	All data plane links in SDN
N	Number of flow groups
f_n	Flow group
s_n	Source switch of f_n
t_n	Destination switch of f_n
$D_b(n)$	Total data plane bandwidth demand of f_n
$D_{fe}(n)$	Total flow table demand of f_n
$D_{sc}(n)$	Total control channel bandwidth demand of f_n
P_n	Path candidates for f_n
p	Path in P_n
$C_b(e)$	Bandwidth capacity of link e
$C_{fe}(v)$	Maximum flow table size of switch v
$C_{sc}(v)$	Control channel bandwidth capacity of v
X_n	Flow table satisfaction ratio of f_n
$\vec{X}$	Flow table satisfaction ratio vector
Y_n	Control channel satisfaction ratio of f_n
$\vec{Y}$	Control channel satisfaction ratio vector
x_n^p	Whether f_n goes through path p
y_e^p	Whether path p goes through link e
z_v^p	Whether path p goes through switch v
δ	Fairness relaxation parameter
R_{fe}, R_{sc}	Unsaturated switches in terms of flow table and control channel
Q_{fe}, Q_{sc}	Non-bottlenecked flows in terms of flow table and control channel
$U_{fe}(v), U_{sc}(v)$	Current utilization level of switch v in terms of the two resources
$T_{fe}(v), T_{sc}(v)$	Total demands of non-bottlenecked flows on v in terms of the two resources
v_{fe}, v_{sc}	The most bottlenecked switch in terms of flow table and control channel

fected by the amount of allocated flow table resource, the changes are smooth. Besides, since our algorithm can run in an online fashion periodically and tolerate certain amount of divergence between the real-time traffic and the inputs as discussed later, the current traffic measurement or previous traffic statistics can be used as input demands.

Simple shortest-path routing may lead to unfavorable cases for resource sharing in the network. Besides, with SDN deployed, optimized routing becomes possible. Thus in this work, we explore the resource allocation problem with routing path selections to improve the optimization results. The number of paths connecting the sources and destinations can

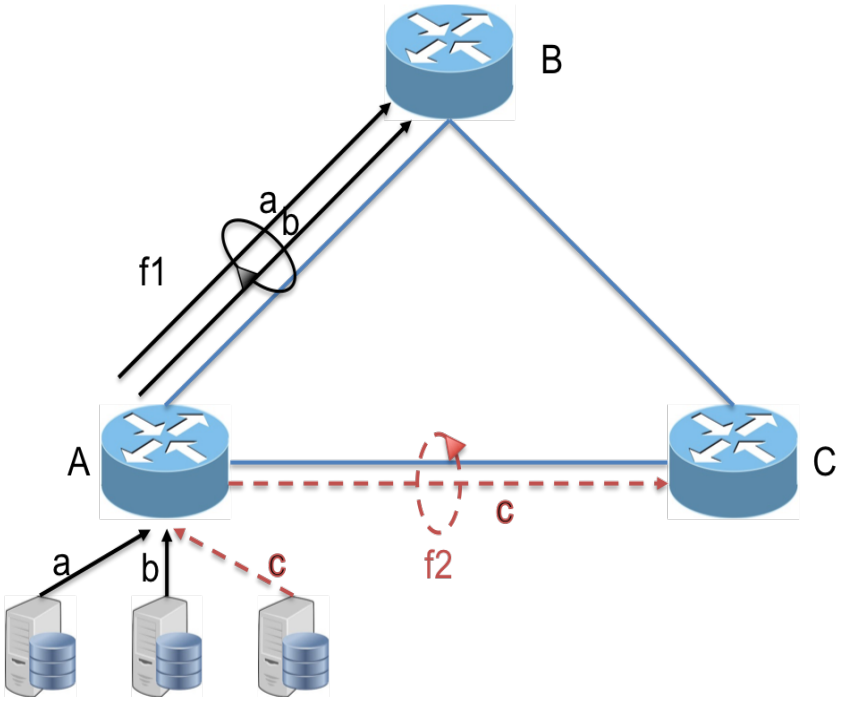


Figure 5.1: Simple network model

be of exponential size in network, but in practice, potential paths for routing the traffic are limited to a specific set and provided ahead of operation time. We thus consider a practical case where a set of pre-generated paths p for flow group f_n are provided as inputs. Let P_n be the set of p for f_n and the size of P_n is k.

Next, we present a simple example as an illustration. Consider the scenario in Fig. 5.1, where three flows a, b and c are aggregated at switch A. Since flow a and b are destined at switch B, flow a and b are aggregated as flow group:

$$f_1 = (A, B, D_b(1), D_{fe}(1), D_{sc}(1)),$$

where $D_b(1), D_{fe}(1)$, and $D_{sc}(1)$ are the aggregated data plane link bandwidth, flow table size, and control channel bandwidth requested by flow a and b. Assuming the number of flow entries required by a and b is 1, if both flow a and b belong to the same QoS class and can be aggregated, $D_{fe}(1) = 1$. On the other hand, if a and b are from different QoS classes, $D_{fe}(1) = 2$. $D_{sc}(1)$ is the control channel bandwidth required to maintain flow a

and b. Similarly flow c is aggregated as flow group:

$$f_2 = (A, C, D_b(2), D_{fe}(2), D_{sc}(2)).$$

Let $k = 2$, which is the maximum number in this case,

$$P_1 = \{(A, B), (A, C, B)\},$$

and

$$P_2 = \{(A, C), (A, B, C)\}.$$

Our algorithm should return the optimal routing paths for f_1 and f_2 on P_1 and P_2, and the amount of resource allocated to each group to meet our objective. A feasible solution should satisfy the following capacity constraints:

- For each flow group f_n, the total resources assigned to it should be equal to or less than its demands.
- For each data plane link e, the total amount of flows going through it should not exceed $C_b(e)$, where $C_b(e)$ is the bandwidth capacity for link e
- For each node v, the total number of flow entries assigned to the flow groups should not exceed $C_{fe}(v)$, where $C_{fe}(v)$ is the total number of flow entries of all flow tables at v.
- For each node v, the total traffic going through its control channel should not exceed its capacity $C_{sc}(v)$.

5.3 Satisfaction Maximization and Fairness

5.3.1 Satisfaction Maximization

Since maximizing the actual amount of resources might lead to unfair solutions where a high-demanded group only gets a small fraction of resources, we ensure the first level of fairness in the allocation problem by targeting on maximizing the ratio between allocation and demand for each group.

The satisfaction ratio of the flow table demand for flow group f_n, $X_n \in [0, 1]$, is defined as the ratio between the allocated flow table space for f_n on the selected path, and its demand $D_{fe}(n)$. If $X_n < 1$, cases will happen where a new flow comes into the network and the allocated flow table resources are already fully used. As introduced in Chapter 2, the switch will evict another entry of the same flow group based on certain eviction rules and install the new entry. However, when the packets from the evicted entry come into the network, they will experience extra delay due to the new flow entry setup process. The satisfaction ratio of control channel bandwidth demand, $Y_n \in [0, 1]$, is defined in a similar way. The satisfaction ratio vectors/resource allocation vectors of the two resources are $\vec{X} = [X_1, X_2, \cdots, X_N]$, and $\vec{Y} = [Y_1, Y_2, \cdots, Y_N]$. The overall satisfaction ratio of a flow group is the sum of the two individual ratio ($X_n + Y_n$). We target on maximizing the total overall satisfaction ratio of all the flow groups. Additional constraints on the value of X_n and Y_n are set to avoid highly unbalanced solutions as discussed in next section. Since the data plane link utilization level is relatively low in DCNs [6], we assume that the data plane bandwidth can always be well satisfied, and so we do not include it in the maximization objective.

Let the variable $x_n^p \in \{0, 1\}$ denote whether flow f_n goes through path $p \in P_n$ or not, and variable $y_e^p, z_v^p \in \{0, 1\}$ denote whether path p goes through link e and node v or not.

The capacity constraints can be presented as:

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p y_e^p \cdot 1 \cdot D_b(n) \leq C_b(e) \qquad \forall e \in E \tag{5.1}$$

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot X_n \cdot D_{fe}(n) \leq C_{fe}(v) \qquad \forall v \in V \tag{5.2}$$

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot Y_n \cdot D_{sc}(n) \leq C_{sc}(v) \qquad \forall v \in V \tag{5.3}$$

$$\sum_{p \in P_n} x_n^p = 1 \qquad \forall n \leq N \tag{5.4}$$

$D_{fe}(n)$ is defined as the number of flow entries required by the f_n, which should be an integer. Since we are considering aggregated flow groups in this work, the value of $D_{fe}(n)$ is at the level of hundreds and the flow table capacity is at the level of thousands. Allowing the value of $X_n D_{fe}(n)$ to be fractional when solving the optimization problem, and rounding it to its nearest integer when deploying the network will not affect the performance significantly, but will improve the computation efficiency by a substantial amount.

We refer to the Pure Maximization case as the optimization problem with the objective to maximize the total satisfaction ratio in the network for all flow groups subject to the above resource constraints. However, since each flow group represents all the flows generated from a single switch, fairness should also be considered to avoid severely biased cases where some flow groups are poorly satisfied in the Pure Maximization problem. In order to address this issue, we consider different fairness models as constraints.

5.3.2 Fairness Models

Simple max-min fairness

We first consider a simple max-min fairness model. The allocation vectors $\vec{X}^*$ and $\vec{Y}^*$ is said to be simple max-min if $min(\vec{X}^*)$ and $min(\vec{Y}^*)$ is maximized among all the possible routings and allocations. Our objective is to seek the maximum satisfaction ratio among all the simple max-min guaranteed allocations. This optimization can be formulated as a mathematical problem composing of two sub-problems: the Max-Min problem to calculate the maximized minimum satisfaction for all flow groups, and the total Satisfaction Maximization problem. The Max-Min problem is defined as:

$$\begin{aligned} \text{maximize} \quad & \alpha + \beta \\ \text{subject to} \quad & Eq(5.1) - Eq(5.4) \\ & X_n \geq \alpha \quad \forall n \leq N \\ & Y_n \geq \beta \quad \forall n \leq N, \end{aligned}$$

and the Satisfaction Maximization problem is:

$$\begin{aligned} \text{maximize} \quad & \sum_{n \leq N} X_n + \sum_{n \leq N} Y_n \\ \text{subject to} \quad & Eq(5.1) - Eq(5.4) \\ & C1 : X_n \geq \delta \cdot \alpha^* \quad \forall n \leq N \\ & C2 : Y_n \geq \delta \cdot \beta^* \quad \forall n \leq N, \end{aligned}$$

where α^* and β^* are the solution to the first sub-problem.

Instead of setting $X_n + Y_n \geq \alpha + \beta$, we tighten the constraints by setting $X_n \geq \alpha$ and $Y_n \geq \beta$. This avoids the highly suboptimal solutions of very large X_n with very small Y_n or vice versa. Constraints $C1$ and $C2$ in the Satisfaction Maximization problem enforce the fairness constraint by making sure that each flow group has a minimum overall satisfaction

ratio at $\delta(\alpha+\beta)^*$. The fairness relaxation parameter $\delta \in [0,1]$ allows controlled trade-offs between satisfaction and fairness. When $\delta = 1$, the solution leads to perfect simple max-min fairness with the minimum satisfaction ratio maximized. The fairness constraint is relaxed as δ decreases and when $\delta = 0$, the optimization problem corresponds to the Pure Maximization case with no fairness constraint.

Classical max-min fairness

The classical max-min fairness is said to be achieved by an allocation if and only if the allocation is feasible and an attempt to increase the allocation of any participant necessarily results in the decrease in the allocation of some other participant with an equal or smaller allocation. The classical max-min fair allocation can be obtained by a progressive filling algorithm where the allocation for each user starts from 0. Since our objective is to achieve fairness in terms of the satisfaction ratio with two resources, we propose a slightly different algorithm than the conventional progressive filling method as described in Algorithm 2. The algorithm seeks classical max-min for the flow table satisfaction ratio and control channel satisfaction ratio separately. The evaluation results show that this method can actually achieve a high and fair allocation for both resources at the same time. In Algorithm 2, parameters R_{fe}, R_{sc} denote the unsaturated switches in terms of flow table and control channel, and Q_{fe}, Q_{sc} denote the non-bottlenecked flows for the two resources. $U_{fe}(v), U_{sc}(v)$ and $T_{fe}(v), T_{sc}(v)$ are the current utilization level and total demands of non-bottlenecked flows on switch v, respectively. Since our optimization objective is the satisfaction ratio, we define the most bottlenecked switch v_{fe}, v_{sc} based on $T_{fe}(v), T_{sc}(v)$, instead of the number of flows on switch v as in the conventional progressive filling algorithm. Besides, the increment of X_n, Y_n is also based on the satisfaction ratio.

Algorithm 2 Classical Max-min Fairness Algorithm with Two Resources

Input: Feasible routing paths assignment x_n^p, traffic demands matrices

Output: $\vec{X}, \vec{Y}$

1: **Initialize** :$\vec{X} \leftarrow \vec{0}, \vec{Y} \leftarrow \vec{0}, R_{fe} \leftarrow V, R_{sc} \leftarrow V$

2: **Initialize**: $Q_{fe} \leftarrow F, Q_{sc} \leftarrow F, U_{fe} \leftarrow \vec{0}, U_{sc} \leftarrow \vec{0}$

3: **Initialize**: $T_{fe}(v) = \sum_{f_n \in Q_{fe}} \sum_{p \in P_n} x_n^p z_v^p D_{fe}(n)$

4: **Initialize**: $T_{sc}(v) = \sum_{f_n \in Q_{sc}} \sum_{p \in P_n} x_n^p z_v^p D_{sc}(n)$

5: **while** $Q_{fe} \neq \emptyset$ or $Q_{sc} \neq \emptyset$ **do**

6: $v_{fe} = \arg\min_v \frac{C_{fe}(v) - U_{fe}(v)}{T_{fe}(v)}, \forall v \in R_{fe}$

7: $v_{sc} = \arg\min_v \frac{C_{sc}(v) - U_{sc}(v)}{T_{sc}(v)}, \forall v \in R_{sc}$

8: $X_n += min(\frac{C_{fe}(v_{fe}) - U_{fe}(v_{fe})}{T_{fe}(v_{fe})}, min_{\forall f_n \in Q_{fe}}(1 - X_n)), \forall f_n \in Q_{fe}$

9: $Y_n += min(\frac{C_{sc}(v_{sc}) - Usc(v_{sc})}{T_{sc}(v_{sc})}, min_{\forall f_n \in Q_{sc}}(1 - Y_n)), \forall f_n \in Q_{sc}$

10: $U_{fe}(v) = \sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot X_n \cdot D_{fe}(n)$

11: $U_{sc}(v) = \sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot Y_n \cdot D_{sc}(n)$

12: $R_{fe} \leftarrow \{v | C_{fe}(v) - U_{fe}(v) > 0\}$

13: $R_{sc} \leftarrow \{v | C_{sc}(v) - U_{sc}(v) > 0\}$

14: $Q_{fe} \leftarrow \{f_n | f_n$ spans only on $v \in R_{fe}$ and $X_n < 1\}$

15: $Q_{sc} \leftarrow \{f_n | f_n$ spans only on $v \in R_{sc}$ and $Y_n < 1\}$

16: **end while**

Furthermore, in our case, the resource allocation is coupled with routing path selection and thus the solution cannot be obtained using only the progressive filling algorithm, where the routing paths should be provided as inputs. To solve this problem, a brute-force method which evaluates all possible routing paths and selects the best one can return the optimal solution, but since brute-force is not scalable in a large network, we propose a heuristic approach to approximate the brute-force method. With a proper initial routing path, the total satisfaction is calculated subject to the classical max-min fairness constraint. Next, the rerouting of the bottlenecked flows is considered. The possible reroutings are evaluated

greedily by starting with the flows at the most bottlenecked switch. To reduce the computation complexity, we only consider rerouting each flow once and set a limit on how many reroutings will be explored in each run. Finally, the routing path selections x_n^{p*} and the allocation vectors $\vec{X}^*, \vec{Y}^*$, which generate the highest total satisfaction ratio with classical max-min fairness, are selected.

To relax the fairness constraint, after we get the optimal routing path selection x_n^{p*} and the allocation vectors $\vec{X}^*, \vec{Y}^*$, we allow the actual satisfaction ratio for f_n (X_n and Y_n) to be in the range of $[\delta X_n^*, \frac{X_n^*}{\delta}]$ and $[\delta Y_n^*, \frac{Y_n^*}{\delta}]$, where $\delta \in [0, 1]$. The maximum satisfaction ratio and the corresponding routing paths can be obtained with the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{n \leq N} X_n + \sum_{n \leq N} Y_n \\ \text{subject to} \quad & Eq(5.1) - Eq(5.4) \\ & \delta X_n^* \leq X_n \leq \frac{X_n^*}{\delta} \quad \forall n \leq N \\ & \delta Y_n^* \leq Y_n \leq \frac{Y_n^*}{\delta} \quad \forall n \leq N. \end{aligned}$$

Equal Share

Another fairness model considered in this work is strictly equal share, where all the flow groups should achieve the same level of satisfaction, which leads to $X_1 = X_2 = \cdots X_N = \alpha$ and $Y_1 = Y_2 = \cdots Y_N = \beta$. This allocation will achieve optimal Jain's fairness index [78]. The maximum satisfaction ratio with equal share fairness guaranteed can be obtained by solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \alpha + \beta \\ \text{subject to} \quad & Eq(5.1) - Eq(5.4) \\ & X_n = \alpha \quad \forall n \leq N \\ & Y_n = \beta \quad \forall n \leq N. \end{aligned}$$

In order to relax the fairness constraint with parameter δ, the following problem needs to be solved:

$$\begin{aligned} \text{maximize} \quad & \sum_{n \leq N} X_n + \sum_{n \leq N} Y_n \\ \text{subject to} \quad & Eq(5.1) - Eq(5.4) \\ & C3 : \delta\alpha^* \leq X_n \leq \frac{\alpha^*}{\delta} \quad \forall n \leq N \\ & C4 : \delta\beta^* \leq Y_n \leq \frac{\beta^*}{\delta} \quad \forall n \leq N, \end{aligned}$$

where α^* and β^* are the solution to the previous problem. Parameter $\delta \in [0, 1]$ allows control of how tightly the solution achieves the fairness objective.

5.4 Performance Evaluation

5.4.1 Simulation Setup

The network topology used for performance evaluation is a fat-tree topology [79] as shown in Fig. 5.2, which has been widely used in DCNs. The topology contains 4 core switches, 8 aggregation switches, and 8 edge switches, which can represent a typical scale of a private data center [6]. Each edge switch is connected to hundreds of servers and the flows are aggregated into groups such that, for each edge switch, there is one flow group destined to every other edge switch. The bandwidth of the data plane links are 10 Gbps and the flow tables of switches are limited to 2000 entries. In this work, we focus on smaller-size DCN and the flow table size is set to match the Broadcom chipset [2] capability, which can be used in smaller DCN, such as private DCN, due to cost control. Newer generations of OpenFlow switches can support larger flow table size and can be used in large enterprise DCN, but in large DCN, the number of flows would be dramatically high and the flow table size is still not sufficient. The same problem exists and needs to be addressed. Multiple connections between switch and controller are proposed in OpenFlow 1.3 and analyzed in [80]. Thus, we assume the control channel bandwidth is 200 Mbps for the core switches, and 100 Mbps for the aggregation and edge switches. The individual demands are gener-

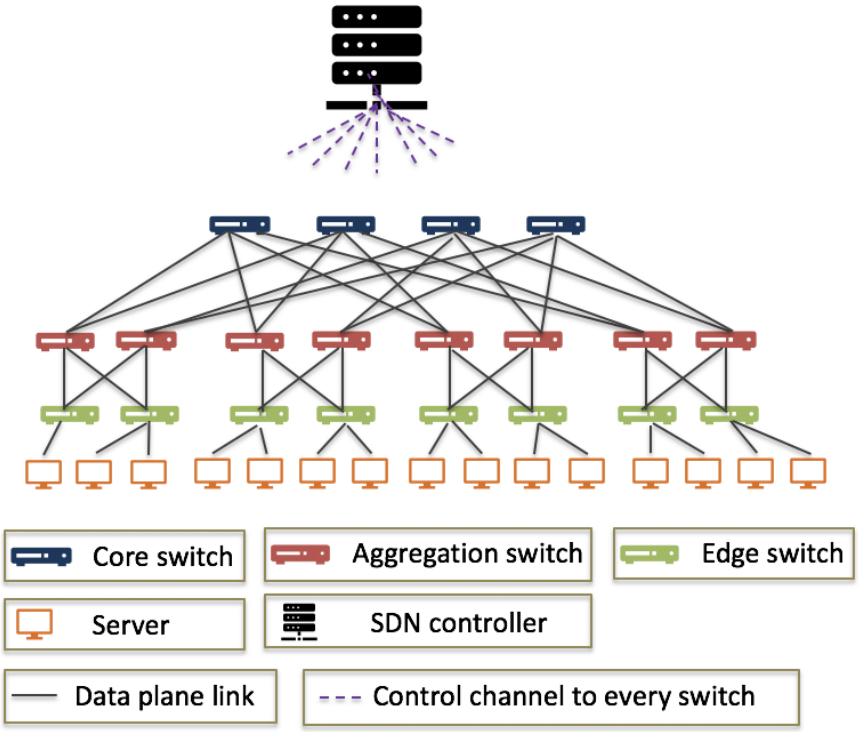


Figure 5.2: Simulation topology

ated randomly according to a uniform distribution and all the results are obtained based on eight random trials.

5.4.2 Running Time

The optimization problems are solved using Gurobi 8.0.1 [81] on a 4-core 3.2GHz Intel i5-6500 processor with 7.7 GB memory and running Linux Mint 17.3. The Gurobi Optimizer is a state-of-the-art solver for mathematical programming. The basic strategy involved is to use a linear-programming based branch-and-bound algorithm. Specifically, the solver first removes all of the integrality restrictions to obtain the linear-programming relaxation of the original problem. Then, a branch-and-bound algorithm is performed to search systematically for the optimal integer solution. The remarkable improvements in solving mixed integer programming have been witnessed in recent years and advance the solver. For example, the advanced preprocessing techniques, which is utilized in the Gurobi solver, can

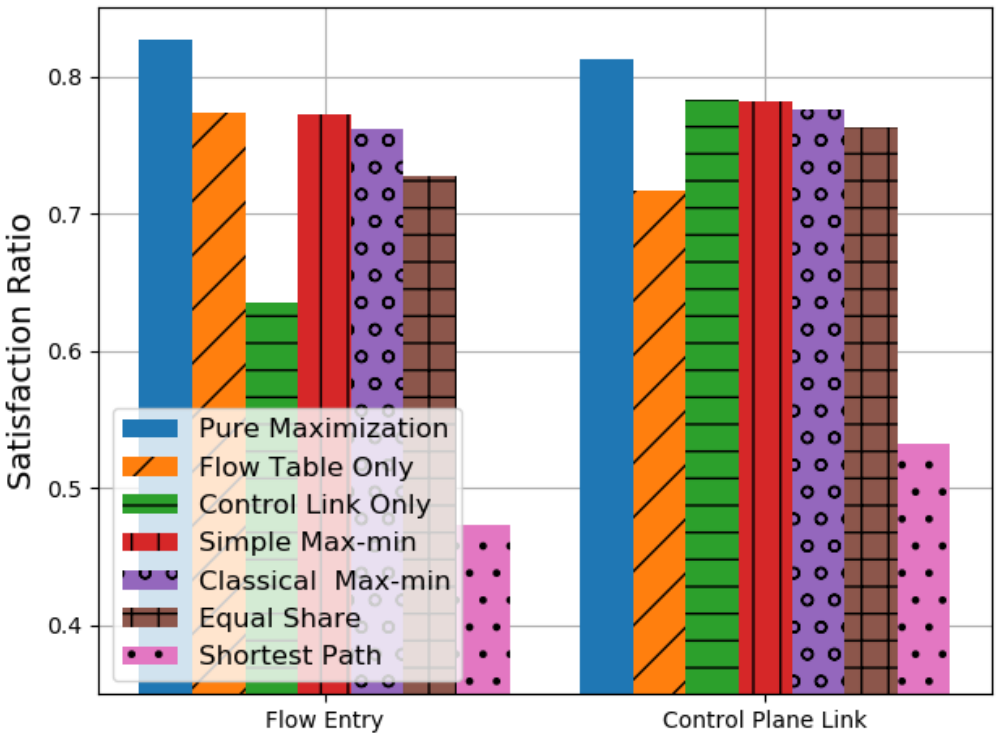


Figure 5.3: Satisfaction ratio comparison among algorithms

be performed before the branch-and-bound algorithm to limit the size of the branch-and-bound tree.

The average running times for the joint optimization cases in Section 5.4.3 with simple max-min and equal share fairness model are $0.93s$ and $0.20s$, respectively. Due to multiple iterations in the classical max-min fairness algorithm, its average running time is $70.2s$. The traffic in a DCN is relatively stable on the timescale of a few seconds up to a few minute [82]. In addition, only the amounts of the three demands need to be collected for calculation, which will incur only a very small overhead in the network. Given these conditions, our algorithm can be run periodically in an online fashion to optimize the network resource allocation.

5.4.3 Comparison of Algorithms

We first compare the satisfaction ratio in terms of the two SDN resources under different optimization scenarios as shown in Fig. 5.3. The total demands for the two SDN resources are set to be larger than the network capacity with the average demand being 200 flow entries and 10 Mbps bandwidth per group. The optimization scenarios include: 1) the Pure Maximization case with no fairness constraint, 2) the optimization cases for a single resource (Flow Table Only or Control Link Only) with simple max-min fairness, 3) the jointly optimized cases for both resources with different fairness constraints, and 4) the optimization case with Shortest Path routing and simple max-min fairness.

First, as expected, the Pure Maximization case achieves the highest satisfaction ratio for both resources. However, it will generate undesirable solutions where some of the flow groups get very low satisfaction ratio. On average, 7% of the flow groups get 0% satisfaction ratio for flow table space and 13% of the flow groups get 0% satisfaction ratio for the control channel resource. To make things worse, the flow groups receiving no flow entry resource are not identical to the flow groups receiving no control plane resource. Thus, some resources are wasted with pure maximization since a flow group cannot be routed successfully with only one SDN resource.

Second, utilizing optimized routing shows substantial improvement compared with the Shortest Path case. Shortest path routing is 30% and 25% worse compared with the optimized routing case with simple max-min fairness enforced, in terms of flow table and control channel satisfaction.

Lastly, when the allocation of a single resource is set as the optimization target (Flow Table Only and Control Link Only), the satisfaction ratio of the targeted resource is slightly better than the corresponding jointly optimized case, but it causes around 10% satisfaction degradation on the other resource. This fact emphasizes the need to jointly optimize the allocation of the two resources. The jointly optimized cases with all three fairness objectives produce well-balanced allocation of the two resources, and achieve almost as good a

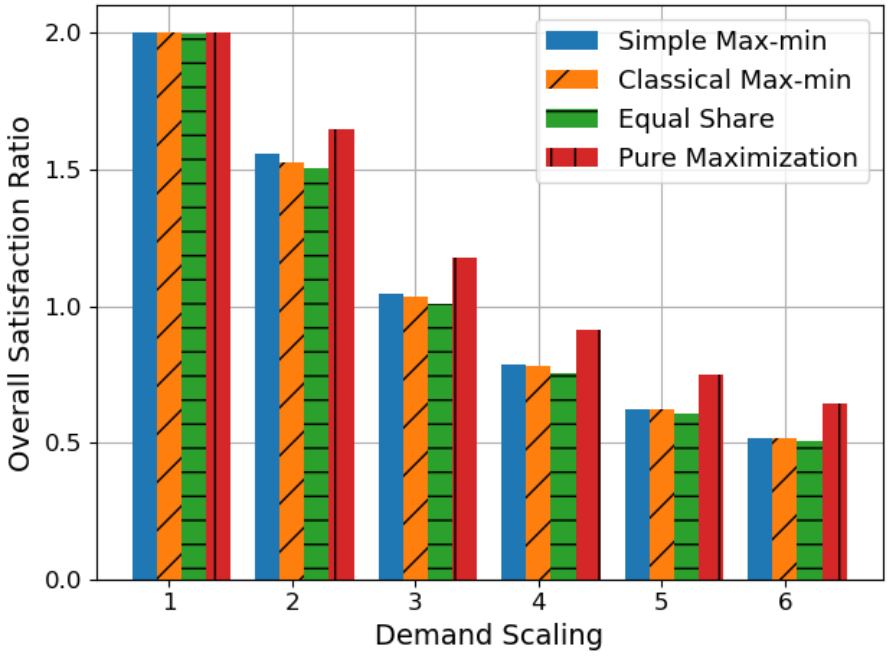
satisfaction ratio as the targeted resource in the single-resource optimization cases.

Next, we will elaborate on the difference between the fairness models and the impacts of fairness relaxation. Since the satisfaction ratios of the two SDN resources can be optimized jointly with all three fairness models as shown in Fig. 5.3, we only show the overall satisfaction ratio (the sum of the two individual ratios) in the following analyses.

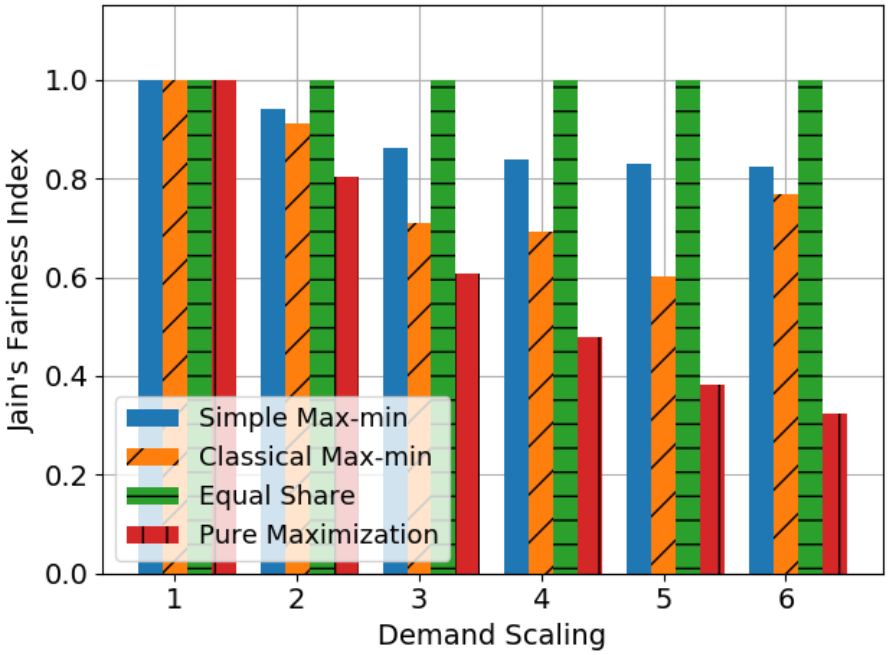
5.4.4 Comparison of Fairness Models

As the traffic in DCNs has grown dramatically in recent years and there is an increasing need to manage the traffic with finer-grained control, the SDN resources might not be sufficient to satisfy the demands of all flow groups. We scale up the demands from where the network resources are sufficient to where resources are insufficient and resource allocation optimization is required, and study the performance using the different fairness models. The overall satisfaction ratios and Jain's fairness index achieved under different demand scalings are shown in Fig. 5.4. A demand scaling of x means x times the baseline demand (100 flow entries and 5 Mbps bandwidth per group on average).

When demand scaling is low, all models achieve the maximum satisfaction ratio of 2.0. As demand scaling increases, the overall satisfaction ratio as shown in Fig. 5.4a decreases, and the pure maximization model achieves higher satisfaction than the other approaches. However, as discussed before, the solution to the pure maximization is undesirable due to unfairness and resource waste. For the other three fairness models, the simple max-min model yields the highest satisfaction ratio while the equal share model generates the lowest at first, but as the demand scaling factor keeps increasing, the benefits of the simple max-min over the classical max-min in terms of the overall satisfaction ratio diminishes and finally, all the three models achieve similar satisfaction ratios. Based on the individual satisfaction ratio of the flow groups, we find that with high demands in the network, the three fairness models tend to generate the similar allocation as discussed in the following subsection.

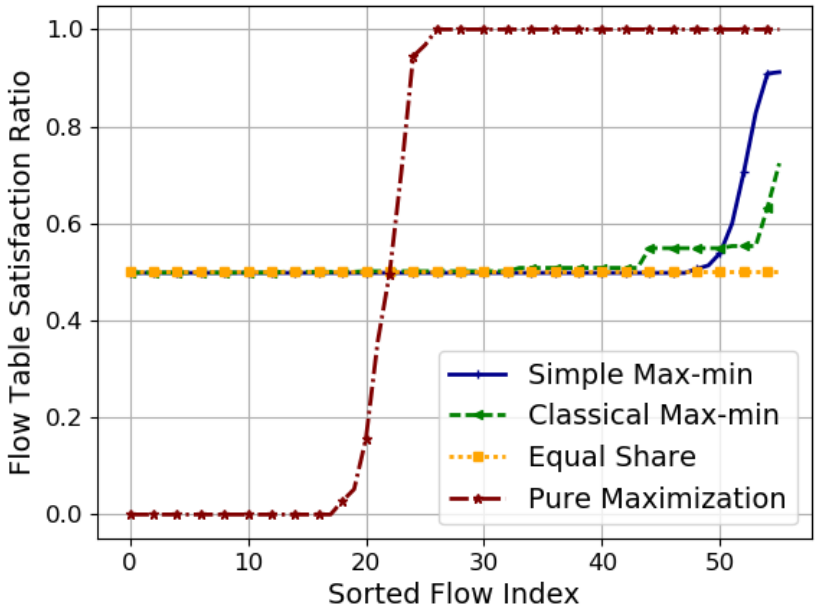


(a) Satisfaction ratio with varying demand scaling

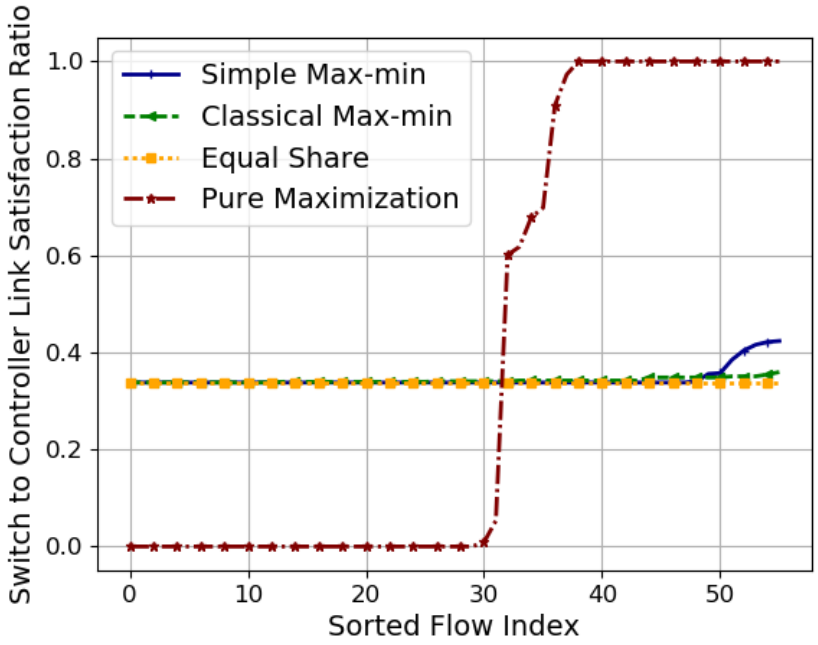


(b) Jain's fairness index with varying demand scaling

Figure 5.4: Algorithm performance with demand scaling



(a) Flow table satisfaction ratio distribution



(b) Control channel link satisfaction ratio distribution

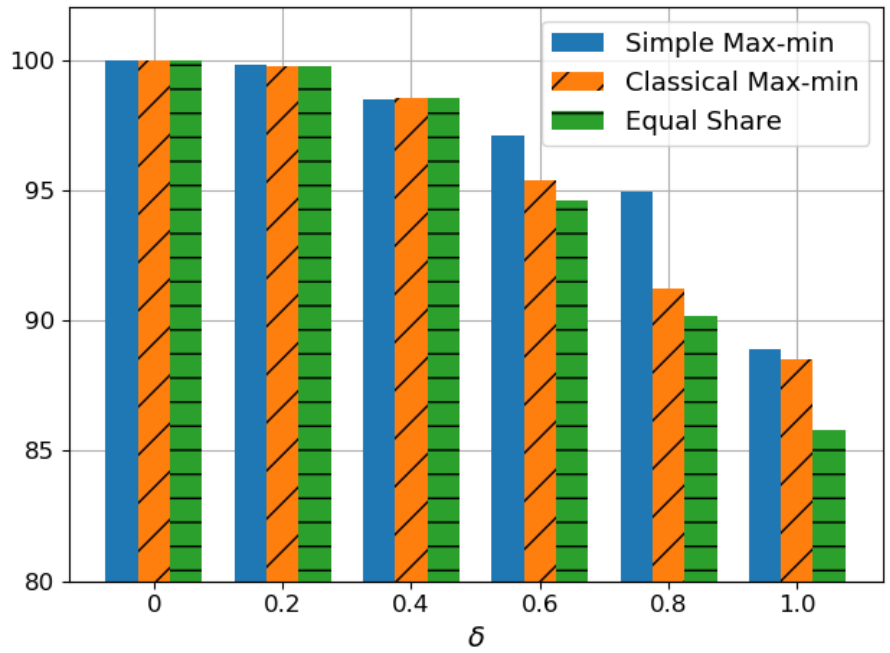
Figure 5.5: Satisfaction ratio distribution

As for the fairness index, when the demand scaling is low, all models achieve perfect fairness since all the flow groups can get the satisfaction ratio at 2.0. As the demand scaling increases, the equal share model still achieves perfect fairness as expected, with the trade-off of low average satisfaction ratio as shown in Fig. 5.4a. The pure maximization model shows a sharp decrease and when the demand scaling factor reaches 6, the fairness index of the pure max algorithm is only around 0.3. The fairness index of the other two models both maintain at a reasonable level with varying demand scaling and the simple max-min model has higher fairness index than the classical max-min model at first. When the demand scaling is large, e.g., the demand scaling is 6, the fairness index of the classic max-min model increases again since different models tend to generate similar allocation with high demands in the network.

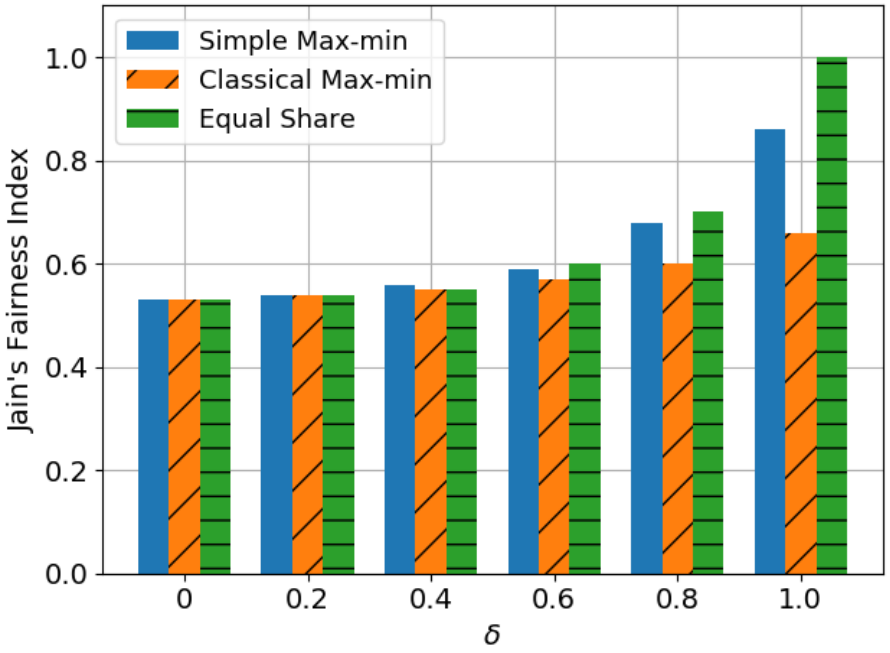
To compare the fairness approaches more closely, we investigate the satisfaction ratios for each flow group when demand scaling is 3 as shown in Fig. 5.5. For both resources, pure maximization causes a severe bias where some groups get 0% satisfaction, which means the access to the network for these flow groups is denied, while other groups get perfect satisfaction. With equal share model, all flow groups obtain equal satisfaction ratio for the two resources as expected. For the two max-min models, the majority of flow groups obtain the same satisfaction ratio as with equal share fairness. A few groups get higher satisfaction than others and thus a higher total satisfaction ratio is achieved than with the equal share model. With the classical max-min model, the flow groups are partitioned to several clusters (usually the number of clusters is very small) and within each cluster, all flow groups achieve exactly the same satisfaction. As the demand scaling increases, the allocation of both the max-min models converges to the equal share model.

5.4.5 Fairness Relaxation

Finally, the impacts of the fairness relaxation parameter δ are studied. When $\delta = 0$, the optimization problems with different fairness objectives all become a pure maximization



(a) Achieved overall satisfaction ratio compared with pure maximization [%]



(b) Jain's fairness index

Figure 5.6: Algorithm performance with varying relaxation parameter

problem. We show the overall satisfaction ratio degradation compared with the pure maximization when demand scaling is 3 with varying δ. The results are shown in Fig. 5.6. As the fairness requirement is tightened, the achieved satisfaction ratio decreases as shown in Fig. 5.6a. When $\delta < 0.6$, the degradation of the three models is the same because the fairness constraint is too relax to take a significant effect among different models. As δ keeps increasing, simple max-min fairness tends to achieve the highest satisfaction ratio. With perfect fairness, the performance degradation compared with the pure maximization algorithm of the simple max-min, classical max-min, and equal share model are 10%, 12%, and 17%, respectively.

We also investigate Jain's fairness index of the satisfaction ratio as displayed in Fig. 5.6b. When the fairness requirement is relaxed ($\delta < 0.6$), the three models all produce the same fairness index. Combined with the satisfaction ratio results, we can infer that with relaxed fairness requirement, the three models produce identical results. As δ increases, while the simple max-min model leads to the highest satisfaction ratio, it also generates a slightly higher fairness index than the classical max-min model. The equal share model achieves the highest fairness index and will reach the optimal Jain's index when $\delta = 1.0$.

In summary, the parameter δ allows control of how tightly the solution achieves the specified fairness objective. This produces a range of solutions that allow for controlled trade-offs between satisfaction and fairness. This would be very useful in scenarios where performance is critical and lower fairness index can be tolerated. Performance and fairness trade-off can be achieved by choosing the proper δ and fairness model. A service provider can determine the best fairness possible for a given minimum performance threshold or determine the best performance for a given minimum fairness threshold.

5.5 Chapter Summary

In this chapter, we address the high satisfaction and fair allocation of resources in software-defined DCNs, including routing path selection. Our approach is to aggregate individual

flows into flow groups and then find the optimal routing paths and the corresponding resource allocation vectors for each flow group. We jointly optimize the allocation of flow table and control channel resources with different fairness constraints. The joint optimization algorithm outperforms the single-objective algorithm and fixed routing algorithm. Three fairness models are compared and a mechanism to relax these constraints is studied. The results show that the simple max-min model is more flexible than the other two and can achieve a better balance in terms of satisfaction ratio and fairness. Besides, the fairness relaxation parameter allows the service provider to control the trade-off between performance and fairness. In next chapter, we will consider the problem of delay-guaranteed fair allocation of resources in software-defined DCN.

CHAPTER 6

DELAY-GUARANTEED FAIR ALLOCATION OF RESOURCES

6.1 Introduction

SDN is proposed to improve the QoS provisioning for various network applications in DCN [83]. For example, the interactive applications (e.g., web search, video call) are delay-sensitive, while the background applications (e.g., data synchronization) are throughput sensitive. SDN can be utilized to provision the network based on their performance requirements. However, the novel architecture of SDN brings new challenges to this problem. Besides the new resource limitations in SDN we addressed in previous chapters, another challenge is that the end-to-end delay in SDN networks is different from the delay in traditional network. When an SDN switch receives a packet, it iterates through its flow tables for matched entries. If the matched entry exists, it will forward the packet to the data plane accordingly. Otherwise, the switch will send a message to the controller for instruction. We refer to the data plane delay as the sum of the entry lookup time and the transmission and propagation delay in the data plane. The control plane delay is regarded as the time the switch used to consult the controller, including the transmission and propagation delay of the packet_in/packet_out message, the controller processing delay, etc. If a matched entry exists, the end-to-end delay of the packet only consists of the data plane delay, which is different from the one in traditional network due to the new entry lookup time. Moreover, if there is no matched entry, the end-to-end delay will involve both the data plane delay and control plane delay. According to the measurements in [84], the delay at SDN switches contributes a large proportion to the entire delay. Therefore, a new expression for the end-to-end delay in SDN considering both the data plane and control plane delay is in need.

Some research works have studied the multi-class traffic management problem in SDN [14, 49, 57] to provide delay guarantee for high priority traffic, such as the interactive traffic, but they are limited in the following aspects. First, they focus on optimizing the data plane bandwidth allocation without considering the resource limitations brought by the SDN architecture. However, in current DCNs, the data plane link utilization level is relatively low [6], and thus the throughput demands of users can be easily satisfied. On the other hand, the resource constraints brought by the unique architecture of SDN are likely to become the new bottleneck. Optimization problems considering the new resource constraints should be explored. Second, the soft or hard delay guarantee offered in [14, 49, 57] only involves the packet transmission and propagation delay on data plane links. In SDN, however, the end-to-end delay must also incorporate delays associated with control functions. Third, the fairness models utilized in [14, 57] obey the strict priorities between traffic classes. Although fairness within a single traffic class is achieved, these models simply favor the allocation for higher priority flows and thus fairness between traffic classes is ignored. Finally, our work [85] in previous chapter studies the fair resource allocation problem with SDN-specific constraints, but delay requirements of flows are not considered.

In this chapter, we formulate the delay-guaranteed fair resource allocation problem in software-defined DCN, with the objective to maximize the total satisfaction ratio of the aforementioned two SDN resources, subject to the fairness and delay constraints. Our approach is to aggregate individual flows into flow groups and then find the optimal routing paths and the corresponding resource allocation vectors for each flow group satisfying our objective. Our main contributions are as follows:

- We address the resource allocation problem coupled with satisfaction maximization, fairness constraint, delay constraint, and routing path selection in software-defined DCNs.

- The end-to-end delay constraint for each packet, considering both the control plane delay and data plane delay in SDN, is derived using queueing theory. To the best of

Table 6.1: Notations for system model in Chapter 6

v	Switch in SDN
e	Data plane link in SDN
V	All switches in SDN
E	All data plane links in SDN
N	Number of flow groups
f_n	Flow group
s_n	Source switch of f_n
t_n	Destination switch of f_n
$D_b(n)$	Total data plane bandwidth demand of f_n
$D_{fe}(n)$	Total flow table demand of f_n
$D_{sc}(n)$	Total control channel bandwidth demand of f_n
P_n	Path candidates for f_n
p	Path in P_n
$C_b(e)$	Bandwidth capacity of link e
$C_{fe}(v)$	Maximum flow table size of switch v
$C_{sc}(v)$	Control channel bandwidth capacity of v
d_n	Worst case end-to-end delay of f_n
d_n^{req}	Delay requirement of f_n
X_n	Flow table satisfaction ratio of f_n
$\vec{X}$	Flow table satisfaction ratio vector
Y_n	Control channel satisfaction ratio of f_n
$\vec{Y}$	Control channel satisfaction ratio vector
x_n^p	Whether f_n goes through path p
y_e^p	Whether path p goes through link e
z_v^p	Whether path p goes through switch v
δ	Fairness relaxation parameter

our knowledge, this is the first work to provide a solution to the resource constrained delay-guarantee problem considering the unique architecture of SDN.

- In order to accommodate various network requirements, we introduce a relaxation parameter $\delta \in [0, 1]$ into the fairness model. It allows the network operator to control the trade-off between total demand satisfaction and fairness.

6.2 Delay-Guaranteed Fair Resources Allocation

In this Section, we first present the system model, followed by the optimization goal. Then the fairness model with a relaxation mechanism is introduced.

6.2.1 System Model

In this work, we consider two broad traffic types:

- Delay-sensitive flows such as those from interactive applications. They take up a small portion of the overall traffic but are highly sensitive to delay – even small delay increases can severely degrade the user experience.
- Non-delay-sensitive flows such as those from background traffic. They have a large bandwidth demand but are insensitive to delay.

The system model follows the model in Chapter 5, but with the new delay consideration. We briefly describe the updated system model here and the parameters used in this chapter are listed in Table 6.1. In the new system model, flow group f_n is defined as:

$$f_n = (s_n, t_n, D_b(n), D_{fe}(n), D_{sc}(n), d_n^{req}),$$

where parameters $s_n, t_n, D_b(n), D_{fe}(n)$ and $D_{sc}(n)$ are defined in previous chapter. d_n^{req} indicates the delay requirement of f_n. Delay-sensitive flows require a bounded end-to-end delay for each packet while the delay requirements for non-delay-sensitive flows are set to ∞ since they are insensitive to delay. As shown in the simple example of Fig. 5.1, the new flow groups f_1 and f_2 becomes:

$$f_1 = (A, B, D_b(1), D_{fe}(1), D_{sc}(1), 100ms),$$

and

$$f_2 = (A, C, D_b(2), D_{fe}(2), D_{sc}(2), \infty).$$

In this chapter, we also explore the resource allocation problem with routing path selections to improve the optimization results. A set of pre-generated possible paths p for each flow group f_n are provided as inputs.

Our algorithm should select the optimal routing path for f_n on P_n, and return the amount of resource allocated to each group to meet the objective. A feasible solution should satisfy the following constraints:

- For each flow group f_n, the resources assigned to it should not exceed its demands.
- For each data plane link e, the total amount of flows going through it should not exceed its bandwidth capacity $C_b(e)$.
- For each node v, the total number of flow entries assigned to the flow groups should not exceed its flow table size $C_{fe}(v)$.
- For each node v, the total traffic going through its control channel should not exceed its capacity $C_{sc}(v)$.
- For each f_n, the worst-case end-to-end delay d_n for each packet does not exceed its delay requirement d_n^{req}.

6.2.2 Satisfaction Maximization with Bounded Delay

The satisfaction ratio of the flow table demand and control channel link demand X_n and Y_n are defined in the same way as in previous chapter. We target on maximizing the total overall satisfaction ratio of all flow groups. The constraints can be expressed as:

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p y_e^p \cdot 1 \cdot D_b(n) \leq C_b(e) \qquad \forall e \in E \tag{6.1}$$

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot X_n \cdot D_{fe}(n) \leq C_{fe}(v) \qquad \forall v \in V \tag{6.2}$$

$$\sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \cdot Y_n \cdot D_{sc}(n) \leq C_{sc}(v) \qquad \forall v \in V \tag{6.3}$$

$$\sum_{p \in P_n} x_n^p = 1 \qquad \forall n \leq N \tag{6.4}$$

$$d_n \leq d_n^{req} \qquad \forall n \leq N \tag{6.5}$$

Constraint Eq (6.1) - Eq (6.4) are the network capacity constraints as defined in Chapter 5. Eq (6.5) defines the new delay constraint.

6.2.3 Priority and Fairness Model

Instead of obeying the strict priorities between different traffic types, we offer a soft priority based on their delay requirements. The delay-sensitive flows will have a higher priority since they have tightener delay requirements. Once the delay requirements of the delay-sensitive flows are satisfied, the resources are allocated fairly to all flow groups.

We use the simple max-min fairness described in previous chapter since it is more flexible than the other two models. This optimization problem can be formulated as a mathematical problem composing of two sub-problems: the Max-Min problem to calculate the maximized minimum satisfaction ratio for all flow groups, and the total Satisfaction Maximization problem. The Max-Min problem is defined as:

$$\begin{aligned} \text{maximize} \quad & \alpha + \beta \\ \text{subject to} \quad & Eq(6.1) - Eq(6.5) \\ & X_n \geq \alpha \quad \forall n \leq N \\ & Y_n \geq \beta \quad \forall n \leq N, \end{aligned}$$

and the Satisfaction Maximization problem is:

$$\begin{aligned} \text{maximize} \quad & \sum_{n \leq N} X_n + \sum_{n \leq N} Y_n \\ \text{subject to} \quad & Eq(6.1) - Eq(6.5) \\ & C1 : X_n \geq \delta \cdot \alpha^* \quad \forall n \leq N \\ & C2 : Y_n \geq \delta \cdot \beta^* \quad \forall n \leq N, \end{aligned}$$

where α^* and β^* are the solution from the first sub-problem, and δ is the fairness relaxation parameter.

6.3 End-to-End Delay in SDN Networks

In order to calculate the end-to-end delay and derive the delay constraints in SDN, we first introduce a general queueing model (Section 6.3.1) as shown in Fig. 6.1 and then we incorporate the system model parameters into it (Section 6.3.2). The parameters used in the queueing model are listed in Table 6.2.

6.3.1 Queueing Model for SDN

For an SDN switch v, node $S0$ collects all the incoming messages. External packets from the data plane arrive at $S0$ at an average rate of λ_S. These packets include packets from directly connected hosts and packets from other switches. $S0$ also collects the messages from the controller, which will be elaborated on later. We assume the average service rate of $S0$ is μ_{S0}. The probability for the data plane packets not being sent to controller, e.g., packets from the matched flows, is $1 - \beta$. These packets will be sent to the data plane via node S with average service rate at μ_S. The messages need to be sent to the controller, e.g., packets from the unmatched flows, will be sent to one of the nodes in $S1$. In real switch implementation, weighted fair queue can be used to assign different percentages of the control channel bandwidth to different QoS classes to satisfy their requirements. In this

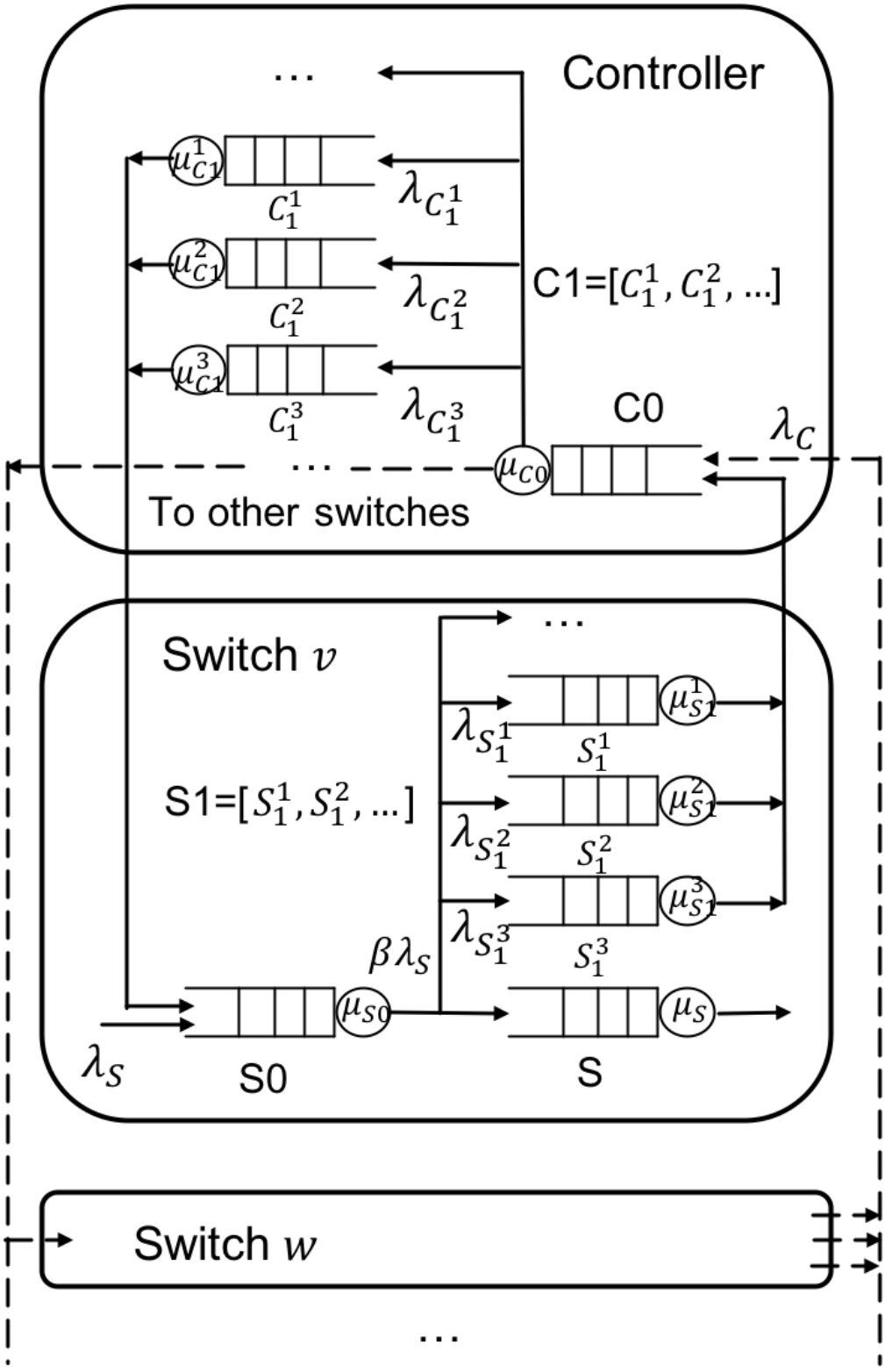


Figure 6.1: Queueing model of SDN.

Table 6.2: Notations for queueing model of SDN

S_0	Queue node at switch to collect arriving packets
S_1	Queue nodes at switch to send packets to controller
$S_1^1, S_1^2, S_1^3, \cdots$	Individual queue node of S_1
S	Queue node at switch to send packets to data plane
C_0	Queue node at controller to collect arriving packets
C_1	Queue nodes at controller to send packets to switch
$C_1^1, C_1^2, C_1^3, \cdots$	Individual queue node of C_1
λ_S	Packet arrival rate to S_0 from data plane
λ_C	Packet arrival rate to C
$\lambda_{S_1^1}, \lambda_{S_1^2}, \lambda_{S_1^3}, \cdots$	Packet arrival rate to $S_1^1, S_1^2, S_1^3, \cdots$
$\lambda_{C_1^1}, \lambda_{C_1^2}, \lambda_{C_1^3}, \cdots$	Packet arrival rate to $C_1^1, C_1^2, C_1^3, \cdots$
μ_{S0}	Service rate of S_0
μ_S	Service rate of S
μ_{C0}	Service rate of C_0
$\mu_{S_1^1}, \mu_{S_1^2}, \mu_{S_1^3}, \cdots$	Service rate of $S_1^1, S_1^2, S_1^3, \cdots$
$\mu_{C_1^1}, \mu_{C_1^2}, \mu_{C_1^3}, \cdots$	Service rate of $C_1^1, C_1^2, C_1^3, \cdots$
β	Probability that the data plane packets sent to controller

work, we model it as the multiple queue nodes in $S1$ with different average service rates $\mu_{S_1^1}, \mu_{S_1^2}, \mu_{S_1^3}, \cdots$. The messages will be sent to the assigned queue node in $S1$ based on their classes. Let $S1 = [S_1^1, S_1^2, S_1^3, \cdots]$ and the corresponding average packet arrival rate is $\lambda_{S_1^1}, \lambda_{S_1^2}, \lambda_{S_1^3}, \cdots$. Other switches (e.g., switch w) in the network will operate in the same way. The packets flow between switch w and the controller is shown as the dashed lines in Fig. 6.1.

For the SDN controller, node $C0$ collects all the incoming messages while the nodes in $C1$ sends messages back to the corresponding switches. We assume the overall packets arrival rate from all switches to controller is λ_C. The average service rate at $C0$ is μ_{C0}. After being processed at $C0$, the incoming packets will be sent back to the switch via one of the nodes in $C1$, with corresponding arrival rate to $C_1^1, C_1^2, C_1^3, \cdots$ being $\lambda_{C_1^1}, \lambda_{C_1^2}, \lambda_{C_1^3}, \cdots$. The service rates of nodes in $C1$ is same as the corresponding nodes in $S1$ since the link between switch and controller is full-duplex. All the packets sent back to the switch will enqueue at $S0$ and then go to node S.

In this model, we assume that the service time of each individual queue follows the

exponential distribution and each host in the network generates packets according to a Poisson process. This queueing model can be analyzed as a Kelly network [86], which is a general multi-class network preserving the Poisson-in-Poisson-out property like Jackson's network [87]. Packets of different QoS classes can be viewed as different types of customers in the network, and will go to different nodes in $S1$ and $C1$. Thus the routing probabilities of different customer types are different. In addition, among the incoming packets to $S0$, all the packets from the control plane will be sent to S while the packets from the data plane will be sent to S with probability $1 - \beta$. This feature makes the network a multi-class queueing network. We know that

- The merging of independent Poisson processes is Poisson with a rate equal to the sum of the individual rates.
- Probabilistic splitting of a Poisson process results in a Poisson process.
- The departure process of an $M/M/1$ queue is Poisson with a rate equal to the input rate of the queue.
- In a Kelly network, the departure process of type i customers from queue k forms a Poisson process.

Combining the aforementioned statements, we can see that the input and output processes of each queue in the network is a Poisson process.

Therefore, the average waiting time for a matched packet at switch v is:

$$\begin{aligned} T_{hit} =& T_{S0} + T_S \\ =& \frac{1}{\mu_{S0} - (1+\beta)\lambda_S} + \frac{1}{\mu_S - \lambda_S}, \end{aligned}$$

where T_{S0} and T_S is the waiting time at queue node $S0$ and S. Meanwhile, assuming the

corresponding node in $S1$ for an unmatched packet is S_1^n, it's average waiting time is:

$$\begin{aligned} T_{miss} =& \frac{1}{\mu_{S0} - (1+\beta)\lambda_S} + \frac{1}{\mu_{S_1^n} - \lambda_{S_1^n}} \\ &+ \frac{1}{\mu_{C0} - \lambda_C} + \frac{1}{\mu_{C_1^n} - \lambda_{C_1^n}} \\ &+ \frac{1}{\mu_{S0} - (1+\beta)\lambda_S} + \frac{1}{\mu_S - \lambda_S} \\ &+ T_{prop}, \end{aligned}$$

where T_{prop} is the two-way propagation delay between the switch and controller.

6.3.2 Queueing Model with System Model Parameters

Next, we incorporate the system model parameters into the queueing model.

λ_S

Assuming the packet arrival rate of f_n follows a Poisson process with rate λ_n. The overall external packets from the data plane arriving at a switch v also follows a Poisson process with rate

$$\lambda_S = \sum_{f_n \in f_v} \lambda_n = \sum_{n \leq N} \sum_{p \in P_n} x_n^p z_v^p \lambda_n,$$

where f_v are the flow groups going through switch v.

β

The value of β is dependent on the flow table satisfaction ratio. Assuming the average length of each flow in flow group f_n is M packets, the probability that a packet in f_n needs to be sent to the controller is:

$$P_{miss}^n = \frac{1 + (1 - X_n)(M-1)}{M}.$$

Meanwhile, the probability that the packet does not need to go to the controller is $P^n_{hit} = 1 - P^n_{miss}$. When $X_n = 1$ (the flow entry demands of f_n is fully satisfied), only the first packet of a flow needs to go to controller ($P^n_{miss} = \frac{1}{M}$). On the other hand, if $X_n = 0$ (the flow entry demands of f_n is not satisfied at all), all the packets of f_n needs to be sent to controller ($P^n_{miss} = 1$). Thus at switch v,

$$\beta\lambda_S = \sum_{f_n \in f_v} P^n_{miss}\lambda_n = \sum_{n \leq N} \sum_{p \in P_n} P^n_{miss} x^p_n z^p_v \lambda_n.$$

λ_C

The overall packet arrival rate to the controller is the sum of the incoming packets from all switches:

$$\lambda_C = \sum_{v \in V} \sum_{n \leq N} \sum_{p \in P_n} P^n_{miss} x^p_n z^p_v \lambda_n$$

μ_S

The average processing rate at queue node S depends on the data plane link bandwidth BW:

$$\mu_S = \frac{BW}{averagePktSize}$$

$S1$ *and* $C1$

Each flow group will get assigned with an individual queue node in $S1$. Assuming flow group f_n is assigned with queue node S^n_1 in $S1$, its average service rate depends on the control channel bandwidth allocated to f_n, where

$$\mu_{S^n_1} = \frac{D_{sc}(n) * Y_n}{averagePktSize}.$$

The service rate at $C1$ of flow group f_n, $\mu_{C_1^n}$, is equal to $\mu_{S_1^n}$ since the link between switch and controller is full-duplex.

$\lambda_{S_1^0}, \lambda_{S_1^1}, \lambda_{S_1^2}, \cdots$

The flow arrival rate to node S_1^n is

$$\lambda_{S_1^n} = P_{miss}^n \lambda_n.$$

Similarly, the flow arrival rate to C_1^n, the queue node assigned to f_n in $C1$, is also $P_{miss}^n \lambda_n$.

6.3.3 Delay Constraint and Approximation

After the first packet of a flow is received and the connection is setup, the end-to-end delay of delay-sensitive flows should be bounded by its requirement. This end-to-end delay constraint is formulated based on the aforementioned queueing model. For the packets following the first packet in an individual flow of f_n, the probability that there is a matched entry in the switch is at least X_n. Thus the worst-case end-to-end delay for each packet in this flow includes 1) the propagation delay from host to source switch and from destination switch to host, 2) the sum of the propagation delay on the passing data plane links, and 3) the worst case waiting time on the passing switches:

$$\begin{aligned} T_{E2E}^n = 2T_{h2s} &+ \sum_{e \in E, p \in P_n} x_n^p y_e^p T_e \\ &+ \sum_{v \in V, p \in P_n} x_n^p z_v^p (X_n T_{hit} + (1 - X_n) T_{miss}), \end{aligned}$$

where T_{h2s} is the propagation delay between hosts and switches and T_e is the propagation delay on the data plane link e. The flows in the same flow group have the same value of worst-case end-to-end delay. Thus we have

$$d_n = T_{E2E}^n,$$

and the delay constraint

$$d_n \leq d_n^{req} \qquad \forall n \leq N$$

can be formulated.

With the delay constraint formulated, we obtain the exact form of our optimization problem. However, this delay constraint is nonlinear and requires complex computation to solve due to the complicated summation terms in both the numerator and denominator. Considering the fact that the traffic in a DCN is relatively stable on the timescale of a few seconds up to a few minute [82], to run our algorithm periodically in an online fashion, we use an approximation form of the delay constraint. The complexity of the constraint exists in the calculation of T_{miss} and T_{hit}. The value of T_{miss} and T_{hit} depends on the traffic load on the controller and each passing switch. Instead of using the exact traffic load, given the traffic demand matrix and the network topology, the overall load the controller and on each switch can be estimated and used. This is done with the following steps:

1. Calculate the average value of X_n and Y_n can be achieved based on the demands and capacities of the flow table and control channel resources.

2. Calculate the average value of λ_S to each switch based on the packet arrival rate of each flow group and the network topology.

3. Given the processing rates of the queue nodes, calculate the value of T_{miss} and T_{hit} based on the results in step 1) and 2).

4. With T_{miss} and T_{hit} calculated, the delay constraint becomes linear and is dependent on X_n and the length of the selected path.

With the approximated delay constraint, we obtain an approximated form of the optimization problem and its exact solution can be obtained in seconds. Our results show that the exact solution to this approximated problem actually also satisfy the exact delay constraints. We will elaborate on these steps and the approximation results in next section.

6.4 Performance Evaluation

6.4.1 Simulation Setup

The network topology used for performance evaluation is same as the one used in previous chapter. In this scenario, each edge switch is connected to hundreds of servers and every edge switch pair has demand in each traffic type. The total demands for the two SDN resources are set to be larger than the network capacity with the average demand being 250 flow entries and 20 Mbps bandwidth per regular flow group, and 50 flow entries and 4 Mbps bandwidth per delay-sensitive flow group. The packet arrival rate for each regular flow group is 600 packets/s while for the delay-sensitive flow group, the number is 200 packets/s. The delay requirements for the delay-sensitive flow groups are generated randomly in the range [100, 150] ms, which are typical values for interactive applications, such as video conferencing. The bandwidth of each data plane link is 10 Gbps and the flow table size is limited to 2000 entries. We assume the control channel bandwidth is 200 Mbps for the core switches, and 100 Mbps for the aggregation and edge switches. The values of μ_{S1}, μ_{C0} and the average flow length M are set to be $100K, 110K$ and 15, respectively, which are in line with existing literature [59, 88, 6].

The optimization problems are solved using Gurobi 8.0.1 on a 4-core 3.2GHz Intel i5-6500 processor with 7.7 GB memory and running Linux Mint 17.3. The average running times is $5.4s$. As mentioned before, the traffic in a DCN is relatively stable on the timescale of a few seconds to a few minutes. Moreover, only the amounts of the three demands are needed for optimization, and collecting that information will incur only a very small overhead in the network. Given these conditions, our algorithm can be run periodically by a centralized processor.

Table 6.3: End-to-end delay comparison $[ms]$

Average $[X_n, Y_n]$	Delay	$\lambda_{Reg}/\lambda_{Delay}$ [packets/s]	
		600/200	300/100
[1, 1]	T_{trad}	4.0	4.0
	T_{E2E}	4.03	4.03
[0.77, 0.77]	T_{trad}	4.0	4.0
	T_{E2E}	7.04	6.9
[0.51, 0.51]	T_{trad}	4.0	4.0
	T_{E2E}	67.6	10.4

6.4.2 End-to-End Delay

We first compare the end-to-end delay calculated by our model, which involves both the control plane delay and the data plane delay in SDN, against a traditional delay model, where only the propagation and transmission delays on the data plane links are considered. The traditional end-to-end delay is formulated as:

$$T^n_{trad} = 2T_{h2s} + \sum_{e\in E, p\in P_n} x^p_n y^p_e T_e + \sum_{v\in V, p\in P_n} x^p_n z^p_v \frac{1}{\mu_S - \lambda_S}.$$

We vary the network load in the scenario and calculate the delay on a 3-hop path using these two models. The results are displayed in Table 6.3. The network load is defined in terms of the average satisfaction ratios of the two SDN resources and the packet arrival rates. The traditional model does not consider the processing time at the switch and controller. Besides, since the data plane links are sufficient for packet transmission, the major component of T^n_{trad} is the propagation delay and it does not vary much with the change of network load. As the network load increases, the results of the two models diverge, indicating that the control plane delay and the entry lookup time dominate the overall delay with higher loads. Moreover, even when the network load is light, the traditional model still cannot precisely capture the end-to-end delay.

After calculating the value of T_{miss} and T_{hit} based on the current network load, we

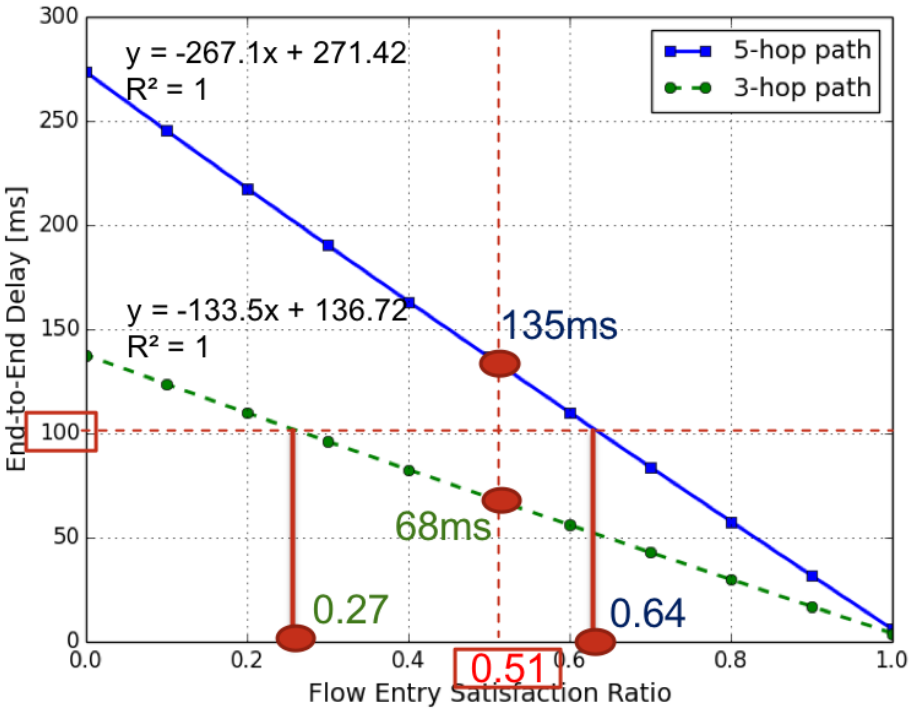


Figure 6.2: End-to-end delay with varying X_n

change the value of X_n assigned to a particular delay-sensitive flow and its worst-case end-to-end delay is shown in Fig. 6.2 (the average $[X_n, Y_n]$ is [0.51, 0.51]). The relation between the end-to-end delay and X_n is linear and can be fit into a linear equation perfectly. The specific expression is dependent on the length of the path and can be used to formulate the delay constraints. The average value of X_n with this network load is 0.51. However, the corresponding delay time with 3-hop and 5-hop path is 68ms and 135ms, respectively, and this might violate the delay requirement of the particular flow. For example, if the delay requirement is 100ms, the minimum required flow entry satisfaction ratio on a 5-hop path and 3-hop path is 0.64 and 0.27, respectively. This linear relation between X_n and T^n_{E2E} can be used for the delay constraints.

Table 6.4: Algorithm performance

	Avg. Sat. Ratio $[X_n + Y_n]$			Jain's Index		
Algorithm	Reg	Delay	Overall	Reg	Delay	Overall
Non-Delay	0.83	0.90	0.87	1.0	0.85	0.88
Delay-Aware	0.82	0.95	0.89	0.99	0.85	0.86
Pure Max	0.70	2.0	1.35	0.50	1.0	0.62

6.4.3 Performance Comparison

In this section, We compare the performance of three optimization algorithms: our proposed delay-aware optimization (Delay-aware), the non-delay aware optimization algorithm (Non-delay), which is the same optimization problem without the end-to-end delay constraints, and the pure maximization algorithm (Pure Max), which ignores both the fairness constraints and delay constraints. The performance of single-resource optimizations are studied and proved to be less efficient than the joint optimization algorithm in previous chapter and thus they are not shown here. The simulation setup in this section and following sections follows the description in Section 6.4.1. We also tested with different network loads and the results show a similar trend.

We first compare the average overall satisfaction ratio and Jain's index (Table 6.4) in terms of the regular flow groups (Reg), delay-sensitive groups (Delay) and all flow groups (Overall). Since the Non-delay-aware algorithm treats all flows equally, it achieves better fairness than the Delay-aware algorithm. While the satisfaction ratio of the Non-delay-aware algorithm is only slightly below the satisfaction ratio of the delay-aware algorithm, we will later see that this comes at the cost of missing delay requirements for the delay-sensitive flows. The Pure Max algorithm ignores the fairness constraints but ends up favoring allocation of the delay-sensitive flows since they have lower SDN resource demands and therefore are easier to satisfy. All demands of the delay-sensitive flows end up being satisfied, but some of the regular flows get very low satisfaction ratio. On average among the Reg flow groups, 40% get 0 satisfaction ratio for the flow table space and 70% get 0 satisfaction ratio for the control channel resource. Our Delay-aware algorithm introduces

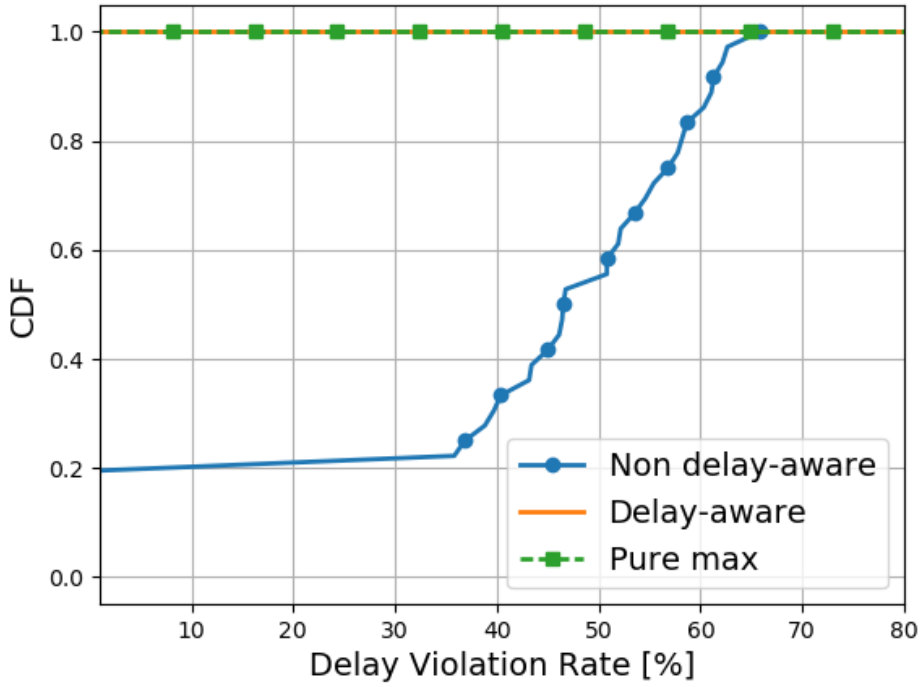


Figure 6.3: End-to-end delay violation from requirements

a balance between the overall fairness and favoring the delay-sensitive flows. It induces a reasonably high satisfaction ratio for the delay-sensitive flows to satisfy their delay requirements. Once the delay requirements are satisfied, the resources are allocated fairly to the remaining flows.

Next, we study the delay violation from the requirements of the delay-sensitive flow groups as shown in Fig. 6.3. The delay violation rate is defined as

$$\frac{max(0, d_n - d_n^{req})}{d_n^{req}} \times 100\%.$$

The *exact* worst case end-to-end delays are calculated based on the path selection and satisfaction ratios returned by the algorithm. Note that Pure Max achieves zero deviation from the requirements. However, as discussed before, Pure Max is undesirable due to its unfairness and resource waste. It is also important to note that, although we used the

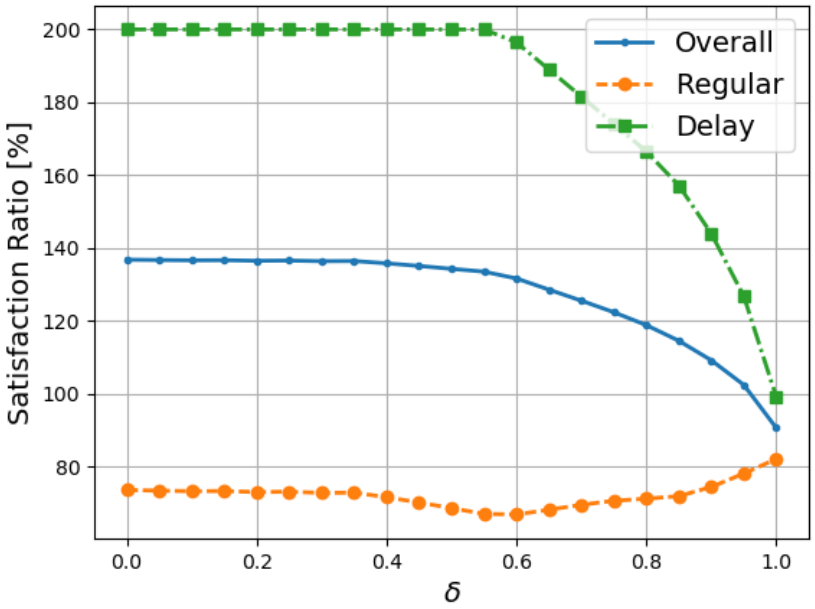
approximated end-to-end delay as constraints in the optimization problem, the exact delay calculated based on the solution satisfies all delay requirements. Finally, without delay awareness, the delay-sensitive flow groups do not meet their delay requirements. Nearly 80% of the delay-sensitive groups exceed their delay requirements by more than 35%.

6.4.4 Fairness Relaxation

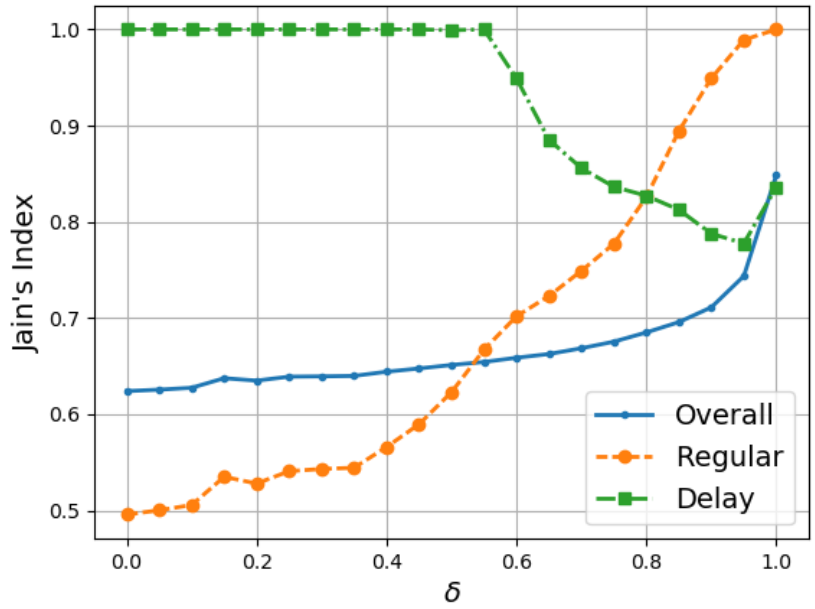
Finally, the impacts of the fairness relaxation parameter δ are studied. We show the overall satisfaction ratio and the Jain's index in terms of the Reg, Delay, and Overall flow groups in Fig. 6.4. In general, the satisfaction ratio of the overall flows decreases as δ increases, and at the same time, the Jain's index of the overall flows increases. Yet the performance trend of each individual traffic type as δ increases is different. When $\delta < 0.6$, the algorithm will favor the allocation for the delay-sensitive flows since their demands are lower than the regular flows. Thus, these delay-sensitive flow groups get a perfect satisfaction ratio of 2.0 while the regular flow groups receive low satisfaction ratios. As the fairness requirement is tightened, the achieved satisfaction ratio for the delay-sensitive flows decreases to even the unfairness between the two flow types. Thus the satisfaction ratios of the two traffic types converge. Regarding the fairness level, Jain's index for the regular flows increases with δ as expected. However, Jain's index for the delay-sensitive flows first decreases then increases as the fairness constraint tightens. This is because in the decreasing phase, more and more delay-sensitive flows get less satisfied while the others are still fully satisfied, which produces an increasing variance and decreasing Jain's index. When $\delta = 1$, all the delay-sensitive flows are satisfied based on their delay requirements. In this second phase, the variance decreases and thus the Jain's index increases.

6.5 Chapter Summary

In this chapter, we address the delay-guaranteed and fair allocation of resources in software-defined DCNs. The delay constraints in the novel SDN architecture are formulated using



(a) Overall satisfaction ratio



(b) Jain's endex

Figure 6.4: Fairness relaxation

queueing theory. We jointly optimize the allocation of flow table and control channel resources with both delay and fairness constraints. A mechanism to relax the fairness constraint is also studied.

CHAPTER 7
PRACTICAL CONSIDERATIONS OF RESOURCES ALLOCATION ALGORITHMS

7.1 Introduction

In Chapter 5 and Chapter 6, we proposed and solved two types of resources allocation problem to better utilize SDN. Before these algorithms can be implemented in a real network, some practical issues need to be considered to make sure the algorithms are both scalable and stable.

As stated before, the optimization problem is NP-hard, which means the problem cannot be solved in polynomial time. The network topology used in previous chapters is a common-size private DCN. Although the running times required for this topology are reasonable, we want to test the complexity of the optimization problem as the network scale increases. Moreover, in order to reduce the complexity of the problem, we allow the number of flow entries assigned to a flow group, $X_n D_{fe}(n)$, to be fractional when solving the problem and round it to its nearest integer when deploying the network. We assume that this rounding step will not affect the performance of the algorithm significantly. However, this assumption needs to be verified. Finally, the algorithm takes traffic demands as inputs and these demands can be obtained from measured statistical traffic characteristics over time or the total amount of resources purchased by the users. Yet the real-time traffic demands when the network is in operation might diverge from the inputs to the algorithm. We want to study how sensitive the algorithm performance is in terms of the real-time demands if they are different from the inputs.

In this chapter, we study some practical issues before the resources allocation algorithms can be implemented in real world. The practical considerations include algorithm

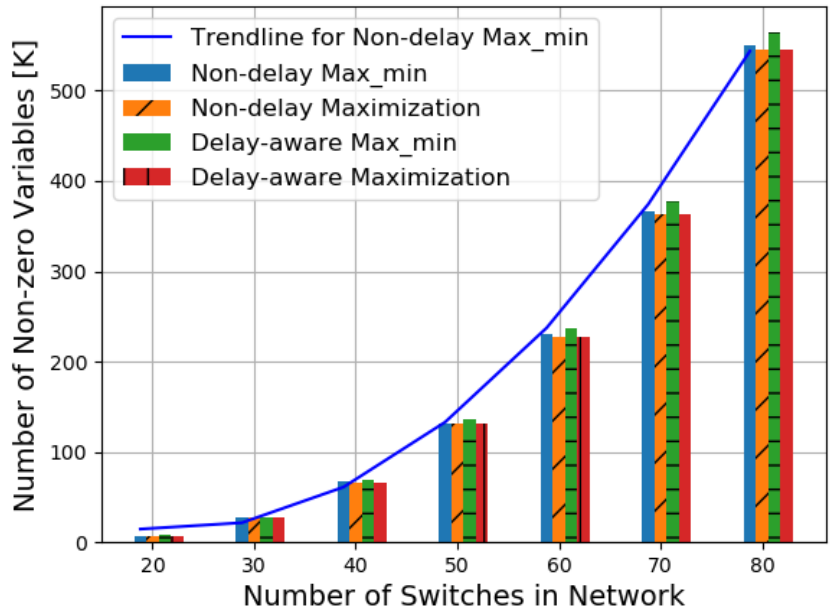
complexity analysis, the rounding effects, and the sensitivity study. The algorithms focused in this work include both the non-delay-aware case (Chapter 5) and delay-aware case (Chapter 6) with the simple max-min fairness enforced.

7.2 Practical Considerations

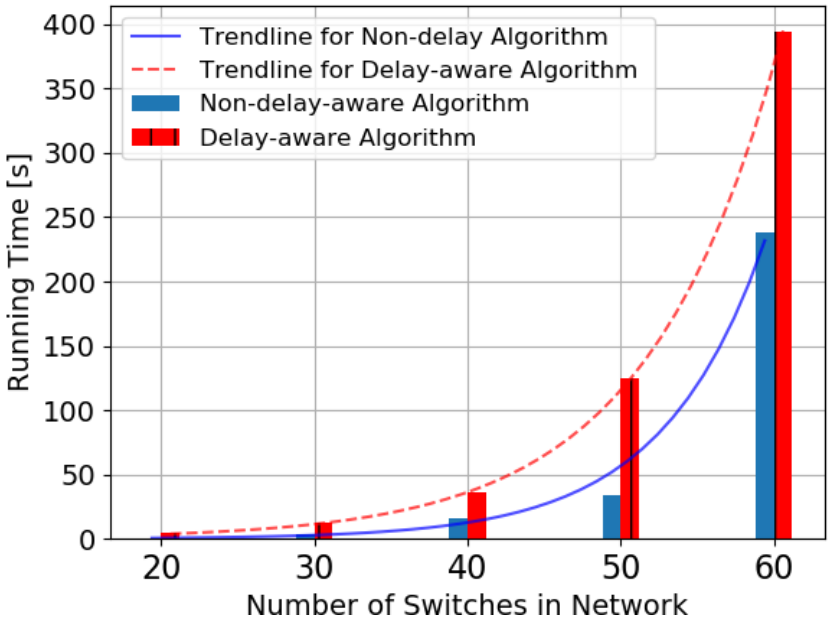
7.2.1 Complexity Analysis

In this section, we scale up the network size and measure both the problem size and the running time. We keep the fat-tree topology since it is one of the most commonly used DCN topologies and increase the total number of switches in the network. More number of switches will incur more number of flow groups in the network. In the simulation, we assume that each edge switch has two flow groups to every other edge switch. For example, in a DCN with 50 switches (10 core switches, 20 aggregation switches, and 20 edge switches), the total number of flow groups is $20 \times 19 \times 2 = 760$. The overall demands of the two resources are set to be larger than the network capabilities. The number of candidate paths, k, also increases with the number of switches to make full use of different paths.

We first examine the number of non-zero variables involved in the optimization problems after the presolve step. Presolve refers to a set of problem reductions that are applied before the branch-and-bound step. These reductions are intended to reduce the size of the problem and to tighten its formulation. This kind of tightening is critical to the solution of an integer program, and makes presolve an important step in the way to get the solution. Therefore, the number of non-zero variables after the presolve step can represent the size of the optimization problem in a more accurate way. Both the non-delay aware algorithm and the delay-aware algorithm are categorized into their two sub-problems: the max-min sub-problem and the satisfaction ratio maximization sub-problem as described in previous chapters. As shown in Fig. 7.1a, the number of non-zero variables increases quadratically with an increasing number of switches in the network. We also display the polynomial-fit



(a) Problem size



(b) Running time

Figure 7.1: Complexity analysis

trendline with order 2 for the max-min sub-problem of the non-delay-aware algorithm. It fits well with the data. The other three trendlines are not displayed to keep the plot concise and clean, but they all fit well with their corresponding data. The R-square values for these trendlines are 0.999, which indicate perfect fitness. There are some other observations. First, the delay-aware algorithm induces more non-zero variables than the non-delay-aware algorithm due to the extra delay constraints. This fact will induce a longer running time for the delay-aware algorithm. Besides, the max-min step induces slightly more non-zero variables than the maximization step.

Next, we measure the running time of both algorithms. All the experiments are carried on a 4-core 3.2GHz Intel i5-6500 processor with 7.7 GB memory. The results are obtained from 8 random simulation runs and are shown in Fig. 7.1b. The running time increases exponentially with the increasing network size. The exponential trendlines fit well with the data, with R-square values being 0.99 and 0.997 for the non-delay-aware algorithm and delay-aware algorithm. The delay-aware algorithm induces longer running time than the non-delay-aware one due to its bigger problem size as shown previously. When there are 50 switches (includes 20 edge switches, 20 aggregation switches, and 10 core switches) in the network, if each edge switches is connected to 50 servers, there are 1000 servers in the network in total, which is bigger than most university DCNs and private DCNs. The running times for this scenario are $32.9s$ and $134.5s$ for the non-delay-aware algorithm and the delay-aware algorithm, respectively. Considering the fact that the traffic in a DCN is relatively stable in a timescale of a few minute [82], the algorithms still can be run in an online fashion in most DCNs.

7.2.2 Rounding Effects

In order to reduce computation complexity, we assume that allowing the value of $X_n D_{fe}(n)$ to be fractional when solving the optimization problem, and rounding it to its nearest integer when deploying the network will not affect the performance significantly. In theory, since

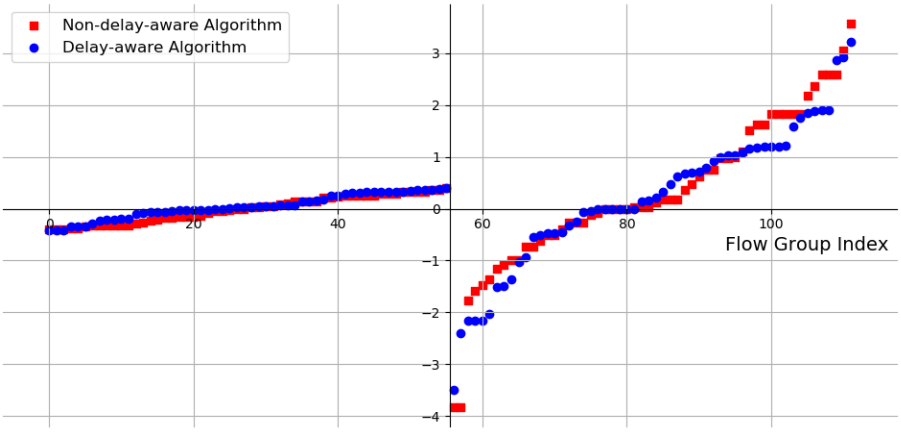


Figure 7.2: Satisfaction ratio variation from theoretical values [%]

the value of $D_{fe}(n)$ in our problem is at the level of hundreds and the flow table capacity is at the level of thousands, this assumption is valid. We still want to study the actual effects of this rounding step in implementation. The simulation scenario in this section and next section follows the description in Section 6.4.1, where the average demand being 250 flow entries and 20 Mbps bandwidth per regular flow group, and 50 flow entries and 4 Mbps bandwidth per delay-sensitive flow group. We also tested with different resource demands/network load and the results show a similar trend.

We first examine the variation between the actual flow table satisfaction ratio achieved in real implementation and the flow table satisfaction ratio calculated from the optimization problem. The satisfaction ratio variation is defined as:

$$\frac{actual\_flow\_table\_sat\_ratio - calculated\_flow\_table\_sat\_ratio}{calculated\_flow\_table\_sat\_ratio} \times 100\%.$$

We display the satisfaction ratio variation for each flow group of the two algorithms in Fig. 7.2. The results for the regular flow groups are shown on the left side of the y-axis and the results for the delay-sensitive flows are shown on the right side of the y-axis. The performance of the two algorithms follow a similar trend and both achieve very small vari-

Table 7.1: Performance variation after rounding

	Satisfaction Ratio		Jain's Index	
	Calculated	Actual	Calculated	Actual
Non-delay-aware Algorithm	0.8667	0.8671	0.8769	0.8766
Delay-aware Algorithm	0.8867	0.8863	0.8638	0.8637

ation. The actual achieved satisfaction ratio is only [-4%, 4%] diverges from the calculated values for both algorithms. The absolute difference in satisfaction ratio is less than 0.02 for the two algorithms. Another observation is that the delay-sensitive flows are more sensitive to the rounding step. While the variation rates for the regular flows are between -0.45% to 0.45%, the values for the delay-sensitive flows are between -4% to 4%. This is due to the fact that the delay-sensitive flows have smaller flow entry demands and rounding the result to the nearest integer would have a bigger impact on them than on flows with larger demands. But overall, the variation rates for the delay-sensitive flows are still very minimal. Finally, we display the calculated satisfaction ratio and Jain's index before the rounding step, and the actual satisfaction ratio and Jain's index for implementation after the rounding step in Table 7.1. The average satisfaction ratio and Jain's index of all flow groups are only slightly affected by this rounding step.

Next, we check if this rounding step will cause flow table over-allocation problem. In average, after rounding the number of flow entries assigned to each flow group to its nearest integer, the flow tables at 15%-25% of the switches are over-allocated only by 0.05%-0.25% for both algorithms, which corresponds to 1-5 entries. Other flow tables are not over-allocated. In this case, we can calculate the over-allocation amount and randomly reduce the number of flow entries allocated to some low-priority flows to make sure no over-allocation will happen in real implementation. Other rounding mechanisms, such as rounding the number of flow entries down to an integer, will avoid the over-allocation problem, but it will cause under-allocation of the resource.

Based on the results discussed above, we come to the conclusion that solving the prob-

lem with fractional numbers for $X_n D_{fe}(n)$ and then rounding it during implementation has minimal effects on the actual performance.

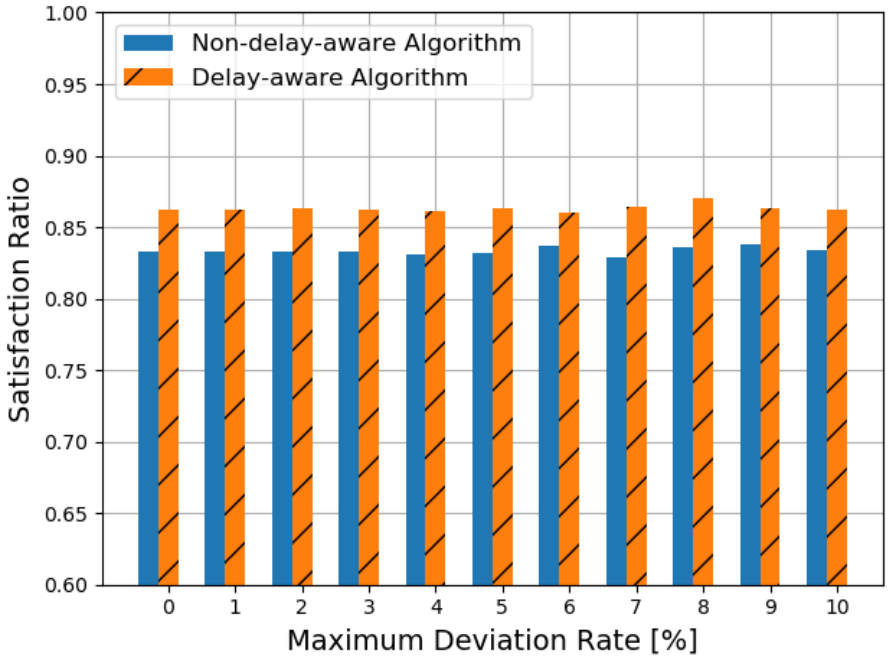
7.2.3 Sensitivity Study

Even though the real-time traffic might be different from the inputs, the allocation to each flow group still follows the solution to the optimization problem. Thus finally, in order to test the performance variation when the real traffic demands are varied from the input demand matrices, we conduct a sensitivity study on the demand matrices. As mentioned in the previous section, the simulation scenario follows the one in Section 6.4.1. After obtaining the optimization solution, we set the actual traffic demands to be deviated from the input demand matrices by a random variation rate. The variation rate is defined as:

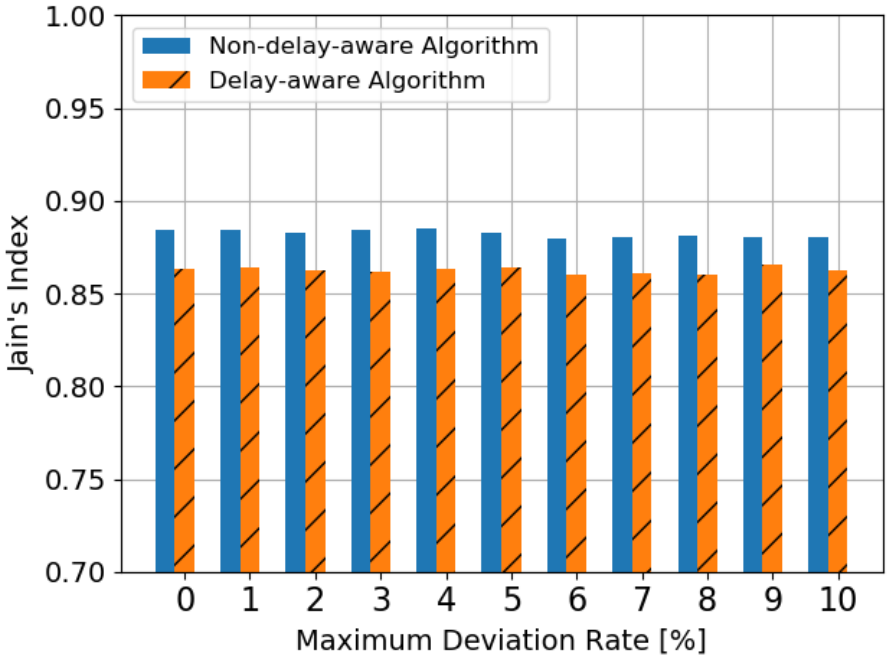
$$\frac{actual\_demand - input\_demand}{input\_demand} \times 100\%.$$

We define a maximum deviation rate and set the random variation rates in the range of [- maximum deviation rate, + maximum deviation rate]. Higher maximum deviation rate indicates larger possible deviations of the actual traffic from the inputs. We tested the algorithms' performance when the maximum deviation rate ranges from 0% to 10%. When the maximum deviation rate exceeds 10%, the traffic in DCN is considered as not "stable" and the algorithms need to run again with the new traffic demands.

We examine the average satisfaction ratio of all flow groups and the Jain's Index as shown in Fig. 7.3. With an increasing maximum deviation rate, both the average satisfaction ratio and the Jain's fairness index stay around the same value. The average value of the satisfaction ratios stays because the real-time demands are randomly distributed and can be both higher and lower than the inputs. The corresponding actual satisfaction ratios can be higher or lower than the calculated value, and thus the average will not change significantly in this case. As for the Jain's fairness index, a maximum deviation rate less than 10% will



(a) Satisfaction ratio



(b) Jain's index

Figure 7.3: Algorithm performance with different maximum deviation rates

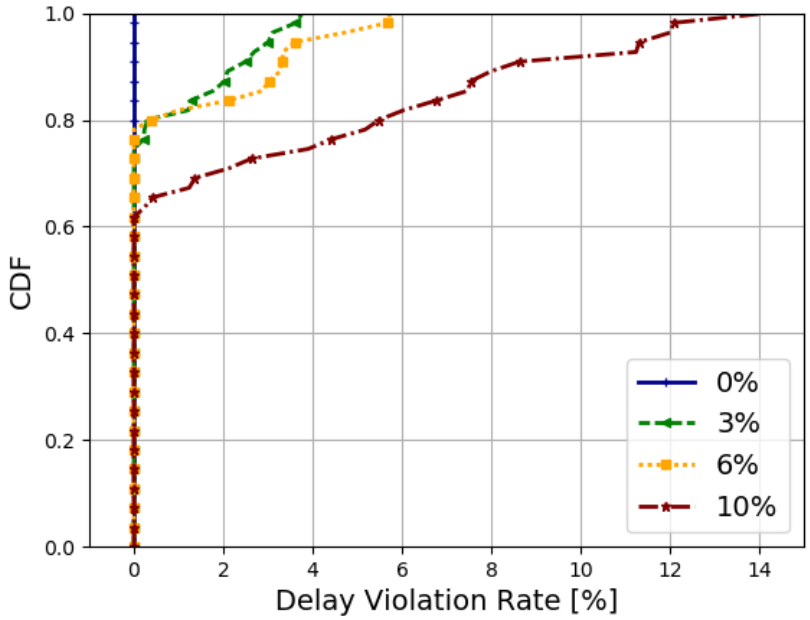


Figure 7.4: Violation from delay constraints with different maximum deviation rates

not cause a very big change in the standard deviation among the satisfaction ratios and thus the fairness index does not vary much. We verify this assumption by increasing the maximum deviation rate to 20%, 30%, and up to 50%. The fairness index experiences an obvious decrease when the maximum deviation rate is large. As stated previously, when the maximum traffic variation is larger than 10%, we consider the traffic as not "stable" and will re-run the algorithm with the new traffic demands. Therefore, the algorithms can provide a stable performance even with varying traffic demands.

Next, we take a look at the delay violation rate with the delay-aware algorithm for the delay-sensitive flows. The delay violation rate is defined as:

$$\frac{max(0, d_n - d_n^{req})}{d_n^{req}} \times 100\%.$$

We can see from Fig. 7.4, when the maximum deviation rate is 0%, which means the real-time traffic is the same as the input, the delay requirements of all flow groups are satisfied. With different maximum deviation rates, there are still around 60% - 80% of the

flow groups satisfying their delay constraints. As the maximum deviation rate increases, the worst-case delay deviation increases. The worst-case deviation for 3%, 6% and 10% maximum deviation rates are 4%, 6%, and 14%, respectively. The delay requirements are set in the range between 100ms to 150ms and the actual delay violation is only 4-15ms.

7.3 Chapter Summary

In this chapter, we study some practical issues for our algorithms to be implemented in real world, namely the algorithm complexity and running time, the rounding effects, and its sensitivity to varying traffic demands. It helps us analyze our algorithms comprehensively. The simulation results show that our algorithms are scalable and can perform stably in a regular-size DCN with varying traffic demands.

CHAPTER 8
CONCLUSIONS

8.1 Conclusions

With the dramatically increasing usage of cloud computing and data center, the traditional network architecture fails to satisfy the emerging requirements. People are turning to SDN for help. However, the unique architecture of SDN brings new challenges to its DCN implementation. As the focus of this dissertation, one of the major challenges is to cope with the new resource constraints brought by the SDN architecture. That is the capacity of the switch-to-controller link, which limits the message exchanges rate between the switch and controller, and the size of the flow table, which restricts the number of flow entries supports by the SDN switch.

In this dissertation, we aimed to optimize network resource allocation for the software-defined DCN in views of the new resource limitations. The contributions in each chapter are summarized as follows:

- In Chapter 3, we developed a queueing model for SDN switches. It considers multiple connections between switches and controller, different interactions between switch and controller, and a limited buffer space at the switch. Validation results are provided to confirm both our mathematical solution and the validity of our model in a real network. The model can provide performance evaluation, including the packet loss rate, the flow setup delay, and the average number of packets in switch, in seconds for realistic DCN traffic loads, which is much faster than the simulation tools. The model can be used to determine the appropriate number of switch-to-controller connections based on the network requirements.

- In Chapter 4, we proposed a lightweight admission control mechanism to determine

if a new flow should be admitted to the network. It makes decisions considering the flow table capacity. This mechanism is implemented at the controller level to follow the basic SDN principle (separation of the control functions and the data components) and take advantage of the flow statistics obtained by other controller functionalities. Simulations are carried out using ns-3 with a canonical DCN topology. The data plane performance is studied to evaluate the improvement brought by the algorithm and the control plane performance is studied to evaluate the algorithm overheads. The proposed mechanism is shown to improve the network performance compared with the default flow table management mechanism while generating tolerable overheads.

- In Chapter 5, we considered a specific problem for allocations of the two aforementioned SDN resources to achieve high satisfaction while maintaining fairness, which can operate in a common-size DCN. We first provided the mathematical formulation of a maximum satisfaction ratio problem with fairness constraints in the software-defined DCN setting. We proposed to aggregate individual flows in the network into flow groups to decrease the load to the flow table and reduce the computation complexity. There are three fairness models considered in this chapter, namely the simple max-min model which requires to maximize the minimum achieved satisfaction ratio, the classical max-min model which means an attempt to increase the allocation of any participant necessarily results in the decrease in the allocation of some other participants with an equal or smaller allocation, and the equal share model where each flow group get the same amount of allocation. The fairness models can be relaxed based on the network requirements using a relaxation parameter δ.

- In Chapter 6, we extended the optimization problem studied in Chapter 5 by considering the delay constraints of each flow. SDN can be leveraged to improve the QoS provisioning for various network applications. Two major types of applications are considered, namely the delay-sensitive applications and the non-delay-sensitive

applications. The end-to-end delay in SDN is derived using queueing theory in view of its novel architecture. We first formulated the exact mathematical problem, then proposed an alternative approximated problem which simplifies the delay constraints to tackle the formulated problem. The alternating approximated problem reduces the complexity of the problem significantly and produces results that actually satisfy the constraints of the original actual problem.

- In Chapter 7, we studied the performance of our proposed algorithm in practical situations. The considerations include the problem size and running time as the network topology scales up, the switch overflow rates and satisfaction ratio variation due to rounding the flow entry allocation to an integer for implementation, and the algorithm performance when the real-time traffic diverges from the inputs in the algorithms.

The theoretical analysis and simulation results provided in this dissertation lay out the foundation of efficient resources allocation in software-defined DCN.

8.2 Future Work

Throughout this dissertation, we focus on optimizing the resource allocations in software-defined DCN. Future research are encouraged in this area. First, the solutions in this dissertation offer a starting point to tackle the challenge, not a set of final solutions.

- In Chapter 3, the analysis and equation derivation is detailed for the single-switch case to let the readers understand the model easily. Further exploration of the multiple-switch network needs to be conducted. Besides, different scheduling algorithms for the multiple switch-to-controller connections can be studied to improve performance.

- In Chapter 4, to improve the performance of the admission control mechanism, intelligent algorithms, such as machine learning techniques, can be studied and utilized to select the algorithm parameters. Implementing the algorithm in data center with multiple controllers can also be explored.

- In Chapter 5 and 6, only one path is selected for each flow group, multi-path selection algorithm can be used to further improve the algorithm performance. Besides, the coping mechanism for a link/node failure should also be studied. Furthermore, a smart mechanism to determine if the algorithm should be run again using the new traffic demands as time goes by for the online implementation is recommended.

Second, the study is limited in scope. Other directions in this area should be investigated to tackle the challenge jointly. Possible directions include enabling multi-tenant resource allocation, network resource utilization monitoring, hardware improvements, etc.

8.3 Publications

As part of the research conducted in this dissertation, we have written several documents that are either published, submitted, or in progress as follows:

- **C. Zhang** and D. Blough, "Resources allocation optimization in software-defined data center networks ," to be submitted.

- **C. Zhang** and D. Blough, "Delay-guaranteed fair allocation of resources in software-defined data center networks ," submitted to IEEE CCNC 2020.

- **C. Zhang** and D. Blough, "High satisfaction and fair allocation of resources in software-defined data center networks," to appear in IEEE ICC 2019.

- **C. Zhang**, H. Yang, G. F. Riley, and D. Blough, "Queueing analysis of auxiliary-connection-enabled switches for software-defined networks," In 2019 International Conference on Computing, Networking and Communications (ICNC) (pp. 497-502). IEEE.

- **C. Zhang**, H. Yang, and G. F. Riley, "Admission control in software-defined data-center network in view of flow table capacity," in IEEE INFOCOM 2018-IEEE Con-

ference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2018.

- H. Yang, **C. Zhang**, and G. Riley, "Support multiple auxiliary TCP/UDP connections in SDN simulations based on ns-3," in Proceedings of the Workshop on ns-3. ACM, 2017, pp. 24-30.

- J. Ivey, H. Yang, **C. Zhang**, and G. Riley, "Comparing a scalable SDN simulation framework built on ns-3 and DCE with existing SDN simulators and emulators," in Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation. ACM, 2016, pp. 153-164.

REFERENCES

[1] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[2] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: A programmable and high performance platform for data center networks.," in *Nsdi*, vol. 11, 2011, pp. 2–2.

[3] *NoviSwitch 2116 High Performance OpenFlow Switch*, `https://noviflow.com/wp-content/uploads/NoviSwitch-2116-Datasheet.pdf`, 2018.

[4] *TCAM - a Deeper Look and the impact of IPv6*, `https://etherealmind.com/tcam-detail-review/`.

[5] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, ACM, 2014, pp. 175–180.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM IMC*, 2010, pp. 267–280.

[7] The Open Networking Foundation, *OpenFlow Switch Specification (version 1.5.1)*, 2015.

[8] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the SDN control traffic in large ISP networks," in *Communications (ICC), IEEE Int'l Conf. on*, IEEE, 2015, pp. 5248–5253.

[9] H. Huang, S. Guo, W. Liang, K. Li, B. Ye, and W. Zhuang, "Near-optimal routing protection for in-band software-defined heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 11, pp. 2918–2934, 2016.

[10] *ns-3 a discrete-event network simulator ns-3.16*, `https://www.nsnam.org/docs/release/3.16/doxygen/index.html`.

[11] Cisco Systems Inc., *Cisco Global Cloud Index: Forecast and Methodology, 2016-2021 White Paper*, 2018.

[12] "Networking in the Era of Virtualization," Nicira, Tech. Rep., 2012.

[13] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 43, 2013, pp. 15–26.

[15] N. Communications, *Ntt communications launches world's largest sd-wan footprint covering over 190 countries with industry's most comprehensive end-to-end sd-wan service portfolio*, Website, `https://www.ntt.com/en/about-us/press-releases/news/article/2017/0620.html`, 2017.

[16] Open Networking Foundation, `https://www.opennetworking.org/`.

[17] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[18] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The switchware active network architecture," *IEEE network*, vol. 12, no. 3, pp. 29–36, 1998.

[19] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in active networks," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 72–78, 1998.

[20] Internet Engineering Task Force, `https://www.ietf.org/`.

[21] T. Lakshman, T Nandagopal, R Ramjee, K Sabnani, and T Woo, "The softrouter architecture," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, Citeseer, vol. 2004, 2004.

[22] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and control element separation (forces) framework," Tech. Rep., 2004.

[23] A. Farrel, J.-P. Vasseur, and J. Ash, "A path computation element (pce)-based architecture," Tech. Rep., 2006.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[25] *Nox resources*, Website, `https://github.com/noxrepo/nox`, 2019.

[26] The Open Networking Foundation, *OpenFlow Switch Specification (version 1.3.0)*, 2012.

[27] D. S. Marcon, F. M. Mazzola, and M. P. Barcellos, "Achieving minimum bandwidth guarantees and work-conservation in large-scale, sdn-based datacenter networks," *Computer Networks*, vol. 127, pp. 109–125, 2017.

[28] H. Owens and A. Durresi, "Explicit routing in software-defined networking (ersdn): Addressing controller scalability," in *2014 17th International Conference on Network-Based Information Systems*, IEEE, 2014, pp. 128–134.

[29] A. Hari, T. Lakshman, and G. Wilfong, "Path switching: Reduced-state flow handling in sdn using path information," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ACM, 2015, p. 36.

[30] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*, IEEE, 2013, pp. 18–25.

[31] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*, ACM, 2012, pp. 7–12.

[32] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Communications*, vol. 11, no. 2, pp. 38–54, 2014.

[33] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[34] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving sdn survivability," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, IEEE, 2014, pp. 1909–1915.

[35] B. Y. Yoon, *Method and apparatus for network failure restoration*, US Patent App. 14/600,892, 2015.

[36] A. Basta, A. Blenk, H. B. Hassine, and W. Kellerer, "Towards a dynamic sdn virtualization layer: Control path migration protocol," in *2015 11th International Conference on Network and Service Management (CNSM)*, IEEE, 2015, pp. 354–359.

[37] *Floodlight project*, Website, `https://groups.io/g/floodlight`, 2019.

[38] *Opendaylight*, Website, `https://www.opendaylight.org/`, 2019.

[39] H. Yang, C. Zhang, and G. Riley, "Support multiple auxiliary tcp/udp connections in sdn simulations based on ns-3," in *Proceedings of the Workshop on ns-3*, ACM, 2017, pp. 24–30.

[40] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with openflow 1.3 support," in *Proceedings of the Workshop on ns-3*, ACM, 2016, pp. 33–40.

[41] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *Passive and Active Measurement*, Springer, 2015, pp. 347–359.

[42] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in openflow networks," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ACM, 2014, pp. 177–188.

[43] H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout master: Dynamic timeout for sdn-based data centers," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, IEEE, 2015, pp. 734–737.

[44] H. Liang, P. Hong, J. Li, and D. Ni, "Effective idle_timeout value for instant messaging in software defined networks," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*, IEEE, 2015, pp. 352–356.

[45] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[46] A. Mimidis, C. Caba, and J. Soler, "Dynamic aggregation of traffic flows in sdn: Applied to backhaul networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, IEEE, 2016, pp. 136–140.

[47] A. V. Akella and K. Xiong, "Quality of service (qos)-guaranteed network resource allocation via software defined networking (sdn)," in *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, IEEE, 2014, pp. 7–13.

[48] W. Hong, K. Wang, and Y.-H. Hsu, "Application-aware resource allocation for sdn-based cloud datacenters," in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, IEEE, 2013, pp. 106–110.

[49] A. Ghosh, S. Ha, E. Crabbe, and J. Rexford, "Scalable multi-class traffic management in data center backbone networks," *IEEE JSAC*, vol. 31, no. 12, pp. 2673–2684, 2013.

[50] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila, "Sdn based inter-technology load balancing leveraged by flow admission control," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, IEEE, 2013, pp. 1–5.

[51] S. Kang and W. Yoon, "Sdn-based resource allocation for heterogeneous lte and wlan multi-radio networks," *The Journal of Supercomputing*, vol. 72, no. 4, pp. 1342–1362, 2016.

[52] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 1734–1742.

[53] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xu, "Dynamic routing for network throughput maximization in software-defined networks," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, IEEE, 2016, pp. 1–9.

[54] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on SDN network throughput," in *Communications (ICC), IEEE Int'l Conf. on*, IEEE, 2017, pp. 1–6.

[55] J. Zhang, D. Zeng, L. Gu, H. Yao, and Y. Fan, "On rule placement for multi-path routing in software-defined networks," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Springer, 2015, pp. 59–71.

[56] T. Feng, J. Bi, and K. Wang, "Joint allocation and scheduling of network resource for multiple control applications in sdn," in *2014 IEEE network operations and management symposium (NOMS)*, IEEE, 2014, pp. 1–7.

[57] J. M. Wang, Y. Wang, X. Dai, and B. Bensaou, "SDN-based multi-class QOS guarantee in inter-data center communications," *IEEE Transactions on Cloud Computing*, 2015.

[58] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "Star: Preventing flow-table overflow in software-defined networks," *Computer Networks*, vol. 125, pp. 15–25, 2017.

[59] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: A network calculus-based approach," in *Global Communications Conference (GLOBECOM), 2013 IEEE*, IEEE, 2013, pp. 1397–1402.

[60] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd international teletraffic congress*, International Teletraffic Congress, 2011, pp. 1–7.

[61] K. Sood, S. Yu, and Y. Xiang, "Performance analysis of software-defined network switch using $M/Geo/1$ model," *IEEE Communications Letters*, vol. 20, no. 12, pp. 2522–2525, 2016.

[62] Y. Goto, H. Masuyama, B. Ng, W. K. Seah, and Y. Takahashi, "Queueing analysis of software defined network with realistic openflow–based switch model," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, IEEE, 2016, pp. 301–306.

[63] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of openflow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172–185, 2016.

[64] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of openflow-based software-defined networks: The multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[65] *OvS: Open vSwitch - Implementation Details*, `http://docs.openvswitch.org/en/latest/faq/design/`.

[66] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[67] G. Latouche, V Ramaswami, and V. Kulkarni, "Introduction to matrix analytic methods in stochastic modeling," *Journal of Applied Mathematics and Stochastic Analysis*, vol. 12, no. 4, pp. 435–436, 1999.

[68] D. John, "Little. a proof for the queuing formula: L= λw," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[69] P. Geraint, K. Vince, Lieke19, L. Sam, caipirginka, T. G. Badger, Nikoleta, C. Alex, and J. Adam, *Ciw: V1.1.5*, 2018.

[70] L. Massoulie and J. W. Roberts, "Bandwidth sharing and admission control for elastic traffic," *Telecommunication systems*, vol. 15, no. 1-2, pp. 185–201, 2000.

[71] A. Kamath, O. Palmon, and S. Plotkin, "Routing and admission control in general topology networks with poisson arrivals," *Journal of Algorithms*, vol. 27, no. 2, pp. 236–258, 1998.

[72] J. Huang, Y. He, Q. Duan, Q. Yang, and W. Wang, "Admission control with flow aggregation for qos provisioning in software-defined network," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, IEEE, 2014, pp. 1182–1186.

[73] J. Leguay, L. Maggi, M. Draief, S. Paris, and S. Chouvardas, "Admission control with online algorithms in sdn," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, IEEE, 2016, pp. 718–721.

[74] S. Qiao, C. Hu, X. Guan, and J. Zou, "Taming the flow table overflow in openflow switch," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, ACM, 2016, pp. 591–592.

[75] B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, "Defending against flow table overloading attack in software-defined networks," *IEEE Transactions on Services Computing*, 2016.

[76] X. Jia, Y. Jiang, Z. Guo, and Z. Wu, "Reducing and balancing flow table entries in software-defined networks," in *Local Computer Networks (LCN), 2016 IEEE 41st Conference on*, IEEE, 2016, pp. 575–578.

[77] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks.," *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[78] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Eesign, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1990.

[79] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 38, 2008, pp. 63–74.

[80] C. Zhang, H. Yang, G. F. Riley, and D. M. Blough, "Queueing analysis of auxiliary-connection-enabled switches for software-defined networks," in *IEEE ICNC*, 2019, pp. 497–502.

[81] Gurobi Optimization, LLC, *Gurobi optimizer reference manual*, 2018.

[82] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 45, 2015, pp. 123–137.

[83] M. Karakus and A. Durresi, "Quality of service in software defined networking: A survey," *Elsevier JNCA*, vol. 80, pp. 200–218, 2017.

[84] T. Zhang and B. Liu, "Exposing end-to-end delay in software-defined networking," *International Journal of Reconfigurable Computing*, vol. 2019, 2019.

[85] C. Zhang and D. Blough, "High satisfaction and fair allocation of resources in software-defined data center networks," in *IEEE ICC*, 2019.

[86] F. P. Kelly, "Networks of queues with customers of different types," *Journal of applied probability*, vol. 12, no. 3, pp. 542–554, 1975.

[87] J. R. Jackson, "Networks of waiting lines," *Operations research*, vol. 5, no. 4, pp. 518–521, 1957.

[88] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ACM, 2013, pp. 13–18.