April 2020

Service Provisioning and Security Design in Software Defined Networks

Mohamed Rahouti
University of South Florida

Service Provisioning and Security Design in Software Defined Networks

by

Mohamed Rahouti

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Electrical Engineering
College of Engineering
University of South Florida

Co-Major Professor: Kaiqi Xiong, Ph.D.
Co-Major Professor: Nasir Ghani, Ph.D.
Yufeng Xin, Ph.D.
Yicheng Tu, Ph.D.
Mahshid R. Naeini, Ph.D.

Date of Approval:
March 27, 2020

Keywords: Quality of Service, Security, Traffic Engineering, OpenFlow, Global Environment for Networking Innovations

Dedication

To my family, my mother Hafida in particular, for all the support, braveness, understanding, and unconditional love.

To my uncle, Bouchta Assemmar, for the unconditional support and love.

To Mr. Mohammed Boutaleb, for the unconditional encouragement and believing in me.

To Dr. Manoug Manougian, for the endless support throughout my Ph.D. studies and summer employment under his supervision.

Acknowledgments

I would like to acknowledge and dedicate my gratitude to the following people who made the completion of this work possible: Most specially, my mother, Hafida Assemmar, my father, Abdelouakil Rahouti, and my siblings, for their unconditional love and support.

My supervisor, Dr. Kaiqi Xiong, for his continuous help and advisement prior and throughout my Ph.D. study.

My supervisor, Dr. Nasir Ghani, for all the endless guidance and support and ensuring my graduate experience and scholarly work are enriched.

My dissertation committee member, Dr. Yufeng Xin, for his contribution and continuous assessment throughout my research.

My research collaborator, Tommy Chin, for his time and the dedicated patience towards training me and ensuring improvement of my technical skills

My dissertation committee members, Dr. Yicheng Tu and Dr. Mahshid R. Naeini, for their valuable suggestions and guidance towards this dissertation work.

God, for giving me strength, capability and perseverance.

Table of Contents

[1] Parts of this chapter were published by the author of this dissertation study in M. Rahouti et al. "LatencySmasher: A Software-Defined Networking-Based Framework for End-to-End Latency Optimization", 44th IEEE Conference on Local Computer Networks (LCN), 2019. The IEEE does not require individuals working on a thesis to obtain a formal reuse license (if they are the senior authors of the published work)

List of Tables

List of Figures

Abstract

Information and Communications Technology (ICT) infrastructures and systems are being widely deployed to support a broad range of users and application scenarios. A key trend here is the emergence of many different "smart" technology paradigms along with an increasingly diverse array of networked sensors, e.g., for smart homes and buildings, intelligent transportation and autonomous systems, emergency response, remote health monitoring and telehealth, etc. As billions of these devices come online, ICT networks are being tasked with transferring increasing volumes of data to support intelligent real-time decision making and management. Indeed, many applications and services will have very stringent Quality of Service (QoS) and security requirements.

In light of the above, effective and secure end-to-end delivery of user data flows is a major focus for network operators. Now in recent years, Software-Defined Networking (SDN) has emerged as a leading communication technology for supporting the evolving service needs of ICT infrastructures. However, even though various efforts have conducted research work, prototype development, and deployment of SDN-based solutions in smaller ICT scenarios, future contributions are still needed. Most notably, there is a lack of cohesive mechanisms for enhancing end-to-end QoS and security for real-time services in SDN systems.

Foremost, stringent delay-sensitive data services, such as emergency response, require effective QoS mechanisms to reduce end-to-end path latency and minimize SDN controller response times. Here, a key concern is how to handle short-term network state fluctuations due to congestion while ensuring latency performance. In addition, security issues, such as large scale Distributed Denial of Service (DDoS) attacks, also pose serious threats to SDN environments. Although various Intrusion Detection and Prevention Systems (IDPS) have been proposed to detect and mitigate such attacks, they often entail significant performance overheads and excessive inspection and/or mitigation times, rendering them impractical.

In light of the above, this dissertation study presents some novel solutions and mechanisms for improving QoS support and security (related to data-control saturation) in SDN-enabled ICT infrastructures. Specifically, an adaptive solution is presented to achieve rapid path computation by leveraging active link latency measurements to generate efficient statistical estimates. Furthermore, a novel priority queueing mechanism is also proposed to improve support for higher-grade services traffic. This solution also integrates and prioritizes control plane traffic to improve overall response and delivery times. Finally, a lightweight kernel-based IDPS scheme is also developed to thwart data-control saturation attacks by leveraging modular string search and filtering techniques. In particular, this solution uses dynamic/self-adjusted detection thresholds to improve attack detection. The proposed methods are all prototyped and tested in the National Science Foundation (NSF) Global Environment for Network Innovations (GENI), a live real-world distributed network testbed facility.

Overall, detailed performance evaluations show that the proposed solutions properly address and resolve many of the research problems outlined in this dissertation study.

Chapter 1 : Introduction

This dissertation studies the performance and security of Software-Defined Networking (SDN) setups in emerging Information and Communications Technology (ICT) infrastructures. Namely, a key focus is on end-to-end delay management for Quality of Service (QoS) support, which is directly affected by SDN controller response time, switching latency, and path computation and selection overheads. In addition, another key focus is the security of such SDN infrastructures against large scale flooding attacks. Hence this initial chapter provides an introduction to some of the main developments in these domains and then introduces the key motivations for this work. The core contributions of this research are then presented in a high-level modality, along with an overview of the rest of this thesis.

1.1 Background Overview

Modern ICT networking systems already form an indispensable part of modern life. Moreover, these infrastructures are constantly being evolved to take on more expansive roles and support an increasingly broad range of emerging "smart" paradigms. Some notable examples here include smart homes and buildings, intelligent transportation and autonomous vehicles, emergency response services, work from home, remote health monitoring and telehealth, etc. Furthermore, a key supporting component here is the emergence and widespread

adoption of smaller, lower cost networked sensor devices for information/data gathering, widely termed as the Internet of Things (IoT). As billions of these devices come online to support many "smart" technology paradigms, traffic volumes are continuing to grow.

Overall, end-to-end QoS support and network security are key requirements for ICT infrastructures carrying an increasingly diverse range of client traffic. Some of the main concerns for networking-enabled services typically include, but are not limited to, routing path computation and selection, QoS support and congestion management, traffic prioritization, disaster response, effective security policies, and targeted attack mitigation. Indeed, modern cloud computing services, IoT paradigms, and other emerging communication technologies open up many new challenges here. These concerns are further compounded by the exponentially increasing heterogeneity and scalability of data services and last mile access technologies. However current enterprise and legacy IP routing technologies (along with their decentralized networking and computing infrastructures) are not well-suited for emerging ICT-based services with stringent real-time data transfers needs. As a result, the technical community is starting to deploy SDN technologies in modern ICT infrastructures in order to improve application-level QoS performance and dynamic service provisioning [1].

1.2 Motivations

ICT networks must provide services support for a broad range of intelligent devices, applications, and systems that are embedded into an ambient environments. Namely, some examples here include sensors integrated in wearable equipment (e.g., such as smart watches),

actuation and automation-enabled devices in smart homes and buildings, vehicular sensors and on-board units for car maintenance and accident avoidance, and so on. These smart devices (along with their associated control systems, automation technologies, and network elements) are all merged together into a networked ecosystem to enable efficient and reliable "smart" application support [2].

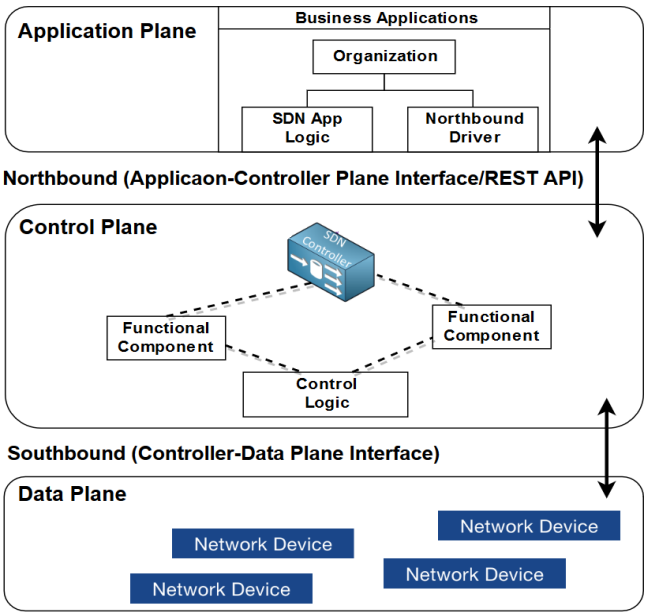

Figure 1.1: A high-level overview of SDN architecture layers.

Now increasingly, many ICT networks are starting to leverage SDN-based technology. This approach uses a centralized controller to separate and consolidate the control plane of a network, allowing streamlined automation of administration, service provisioning, and security policies. In particular, SDN controllers can be programmed to support a wide range of customized applications and make use of the standardized Representational State Transfer (REST) Application Programming Interface (API). The overall SDN architecture is depicted

in Figure 1.1, including the main planes/layers and their functionalities. In particular, the three planes are defined here:

- **Control Plane**: Centralized control structure that embodies a controller and a Network Operating System (NOS). This layer provides hardware-based abstractions for SDN applications as well as a holistic view of the entire network [3], [4].
- **Data Plane**: Termed as the infrastructure or forwarding layer and consists of integrated forwarding elements/components, i.e., switches. Each component maintains a set of rules for directing networking traffic according to the instructions received from the control plane [3], [4].
- **Application Plane**: Set of SDN-based applications that support different customized operations and functionalities, including, but not limited to, QoS support, network security and policy services, as well as implementations [3], [4].

Now unlike legacy routing networks, SDN uses software modules (running at the controller) to generate and place rules for handling data at the switching devices, i.e., instead of using distributed routing protocols. This capability can provide more streamlined and rapid run-time distribution of traffic forwarding rules and security policies. Specifically, SDN controllers operate with a logically centralized global view of network resources. This information can be leveraged to implement customized provisioning strategies and further coupled with programmable interfaces to insert and push forwarding rules to switches via southbound protocols, e.g., such as OpenFlow [5], [6], OpenContrail [7], and Extensible Mes-

saging and Presence Protocol (XMPP) [8]. As such, the "programmability" of the network infrastructure layer in SDN provides a dynamic and cost-effective configuration management solution for ICT services. For instance, SDN can be used to control and regulate IoT systems by expanding connectivity to smart homes using capacity sharing [9]. Alternatively, SDN can be used to assure security for routing devices [2] or improve mobility support in clouds [10], [11].

To date, many studies have looked at the application of SDN in various ICT applications. For example, some efforts have focused on QoS latency and response time management, e.g., [12], [13], [14], [15], [16]. Other studies have also tried to improve the performance of switching devices using various optimization and TE approaches, such as [17], [18], [19]. Nevertheless, despite these contributions, various open challenges still remain here. For example, there is a further need to leverage the real-time data collection/processing capabilities at SDN controllers to achieve more effective, dynamic path computation. This is of particular importance to real-time traffic flows with tight delay QoS bounds (which can easily be impacted by time-varying bursty traffic). Furthermore, there is also a need to support priority queueing for multiple traffic levels in SDN networks. These data plane traffic priorities also need to be properly integrated with SDN control plane traffic.

However, the use of SDN-based technologies also introduces many new attack vectors for malicious actors [20]. Some key examples here include DDoS attacks [21], Link Discovery Service (LDS) exploitation [20], [22], and Man-in-the-Middle (MITM) attacks [20]. Indeed

many of these exploits have already been demonstrated in real SDN settings. Accordingly, various efforts have also looked at improving the security of SDN operation. Of particular concern are large scale DoS flooding attacks, which can be used to overwhelm both the data and control planes in SDN. For example, the SLICOTS [23] and OPERETTA [24] solutions try to thwart Transmission Control Protocol (TCP) SYN flooding attacks by surveiling failed SYN requests. However, there is a further need to develop more effective DoS security solutions here. In particular, these methods should leverage the real-time traffic monitoring capabilities of SDN controllers to implement more dynamic schemes for DoS/flooding detection and mitigation.

1.3 Problem Statement

This dissertation addresses the above challenges and presents a set of novel solutions for improving QoS and security support for SDN-based user services. Specifically, these methods include intelligent Traffic Engineering (TE) path selection as well as priority-based queueing schemes to improve end-to-end service latency. In addition, dynamic IDPS security mechanisms are also proposed to mitigate flooding saturation threats in SDN-based infrastructures.

1.4 Proposed Work and Contributions

The key contributions of this dissertation effort include the following:

1) **End-to-end latency management**: A novel framework is proposed to manage end-to-end latency by leveraging globalized control and topological views at the SDN con-

troller. Specifically, these include latencies associated with path search and selection and controller-switch communication.

2) **Priority-queueing support**: A novel priority queueing mechanism is introduced to improve service support for higher-grade traffic flows. In particular, this solution also integrates control plane traffic to improve overall response and delivery times.

3) **Saturation and flooding attack mitigation**: A novel kernel-based IDPS scheme is proposed using a dynamic threshold-based strategy for detecting TCP SYN flood attacks on SDN infrastructures. This solution improves controller response times and reduces traffic processing overheads.

The rest of this dissertation is organized as follows. First, Chapter 2 presents a survey of the latest research work on QoS and security support in SDN infrastructures. Next, Chapter 3 presents a novel scheme for end-to-end latency reduction in SDN environments. Building upon this, Chapter 4 details a priority-based solution for SDN traffic management. Finally, Chapter 5 addresses the critical issue of SDN security and details a dynamic threshold-based countermeasures solution for detecting and mitigating TCP SYN flooding attacks. Finally, conclusions and future research directions are presented in Chapter 6.

Chapter 2 : Survey of Related Work

The overall topic of QoS support in SDN-enabled communication systems has received a significant amount of attention in the last decade, specifically due to the contemporary nature of this technology and its potential challenges. Along these lines, this chapter presents an overview of some of the latest developments and state-of-art efforts in this particular domain, i.e., including work on latency management, controller overhead reduction, priority-aware flow forwarding, and flooding and data-control plane saturation issues. A broad range of associated mechanisms and provisioning schemes are also surveyed. Lastly, some key open research challenges are summarized to motivate this dissertation study.

2.1 QoS Guarantees in SDN Communication Systems

By one definition, QoS is the qualitative measurement of the overall performance of networking-enabled services, e.g., based upon metrics such as bandwidth throughput, end-to-end communication delays, loss, etc. [12]. Now as noted in Chapter 1, delivering QoS support in SDN-based environments is of a key importance for many users in emerging "smart" technology paradigms [25]. Here SDN provides an effective framework for expediting traffic management and control in emerging data networking environments. Furthermore, SDN also presents a global view of network resources and implements centralized control

provisioning by decoupling network control from the data plane [12]. Hence SDN controllers can simply "program" forwarding rules by pushing them over their southbound interfaces to SDN-enabled switches, e.g., such as Open vSwitch [6]. These capabilities allow network operators to achieve a holistic view of user data flows and build tailored management applications for their clients, e.g., such as TE routing [12], QoS-aware flow routing [26] [27], flow inspection [28], Science De-Militarized Zone (DMZ), and even full in-line Intrusion Detection and Prevention System (IDPS). Namely, the abstraction of the network infrastructure allows networking administrators to program new features and policies through a standard open API without having to adjust their physical infrastructure plane.

Now many high-end users have stringent QoS delays bound requirements, mandating effective management of associated control and switching plane latencies (in SDN settings). Hence various research studies have been carried out to explore and enhance QoS support in SDN networks. Furthermore, the majority of these earlier studies have focused on addressing data plane and packet sojourn times [12], [29]. Carefully note that the data-control communication delay in SDN networks (e.g., time between OpenFlow $Packet\_In$ and $Packet\_Out$ messages) has a significant impact on traffic delivery performance. Indeed, this delay is inevitable between the data and control planes given the inherently centralized nature of SDN operations (and this represents an additional factor in latency increase).

Furthermore, SDN-based switching devices, e.g., such as Open vSwitch platforms, process packets/flows according to the rules injected by the SDN controller through its

southbound interface. Namely, these flow rules are inserted by the SDN controller into data plane nodes in either a proactive or reactive manner. Specifically, in the former approach, the controller populates the traffic rules *prior* the arrival of traffic to the forwarding switch device. Meanwhile in the latter approach, the controller injects and adjusts flow rules dynamically and in real-time. Expectedly, flooding attacks causing data-control saturation (e.g., TCP SYN floods) can complicate reactive flow rule insertion (discussed in Section 2.4).

2.2 Latency and Response Time Management

In the past, various efforts have also looked at the performance of SDN OpenFlow devices. However, most of these studies have not presented efficient mechanisms to manage controller response delays at the control plane. For example, Curtis et al. [13] demonstrated that the collection rate of SDN topology statistics is strictly limited by dimensionality of flow tables. This study also confirmed that collecting such measurements can significantly impact the update rate of flow rules. Accordingly, the authors tabled a solution to reduce the impact the of overheads associated with flow rule matching and processing. Meanwhile, other studies such as Huang et al. [30] and Rotsos et al. [31] also presented an in-depth look at the performance of OpenFlow switches across a broad range of vendor devices.

Other researchers have also focused on efficient resource allocation in SDN setups. For example, Egilmez et al. [14] presented a solution to improve QoS support in OpenFlow-based networks by guaranteeing end-to-end bandwidth allocation for multimedia streaming. Additionally, Sharma et al. [15] also introduced an SDN-based QoS framework leveraging

the capabilities of the Floodlight SDN controller [32]. Namely, this scheme assigns one SDN controller per Autonomous System (AS) routing domain and uses it to communicate through a northbound interface to handle OpenFlow policies (while guaranteeing efficient bandwidth allocation between end-user hosts). Also, Celenlioglu and Mantar [33] presented a framework for routing and resource allocation/management while considering initial routes in SDN-enabled intra-domain settings. This solution was shown to enhance overall routing scalability and improve QoS support.

Furthermore, Jinyao et al. [34] also presented HiQoS, a multi-path mechanism to ensure QoS guarantees in SDN environments. Namely, OpenFlow-based queuing mechanisms were applied to implement bandwidth management and allocation for different types of networking traffic. Multi-path selection was also done here based upon a modified version of the Dijkstra algorithm with QoS constraints. Similarly, Tariq and Bassiouni [35] proposed QAMO-SDN, a QoS-aware solution that implemented multi-path routing in smaller SDN-enabled data center environments. Namely, the proposed solution pre-computed multiple paths between two end-user hosts using the Dijkstra algorithm, and the authors tested their solution in a simulated networking environment.

Similarly, Huang et al. [36] presented another SDN-based multi-path framework for GridFTP (big data transfer protocol). Again, this solution leveraged the Dijkstra algorithm for path selection. Meanwhile, Hussain et al. [37] also assessed a hash-based mechanism for multi-path selection in SDN networks using the Floodlight controller. Specifically, this

solution was designed to schedule different flows by using a hash function to forward them in a balanced manner over pre-computed routes. Finally, Basit et al. [16] detailed a cross-layer coordination scheme between different Internet Service Providers (ISPs) running SDN setups, i.e., peering across multiple Internet Exchange Points (IXPs). This solution focused on improving network throughput and resource efficiency by computing all available end-to-end paths between any pair of nodes. Note that this work assumed the availability of OpenFlow switches with multi-queueing support.

However, despite these many contributions, none of the above studies have tried to leverage real-world time series (TS) data to further improve path selection. Although some TS-based estimation approaches have been proposed [38], they have not been implemented for a per-link latency calculation and QoS-aware path selection in SDN environments.

2.3 SDN-Based Priority Queueing

Rapid response time is a critical requirement for many high-grade user services. In particular, timely communications play a vital role in emergency response services, i.e., since it can directly impact human life and property [39], [40]. However, network resource coordination and decision making can be very challenging under such circumstances [41]. Namely, fast response times can be difficult to achieve and hence associated information infrastructures (providing decision support) must be tailored for highly-responsive operation [42]. At the same time, network operators must maintain existing Service Level Agreements (SLAs) in order to prevent sizable revenue losses [43], [44]. As a result, customers are

usually assigned different traffic priority levels [41] during emergency situations, and traffic processing decisions are made accordingly [45]. However, most existing emergency response communication setups are ill-equipped to handle such requirements, i.e., since they are largely built using older distributed routing protocol technologies. It is here that SDN-based systems offer much potential.

Now many research studies in the past have proposed TE-based solutions, e.g., to improve QoS (for data traffic flows with different priority levels), improve capacity utilization, and minimize flow processing delays and controller response times [46]. For example, Google [47] has applied SDN capabilities to improve capacity utilization between its data centers by using application-specific priority levels. Egilmez, et al. [14] also proposed a SDN-based scheme that used dynamic routing to prioritize multimedia traffic over normal flows, whereas Rahouti, et al. [26] presented a SDN framework for emergency response traffic. Nevertheless, even though SDN offers a broad range of services via its control plane, many of these solutions do not support multi-priority data plane flows or prioritize control plane traffic (by default) [48].

Furthermore, priority queueing and load balancing mechanisms for OpenFlow devices have also been studied, i.e., to support a broad range of QoS-based services and deliver preferential management for certain types of data plane flows [49]. Specifically, several studies have looked at reinforcing switching devices by using optimization techniques [15], [19], [50]. However, these efforts do not handle flows with differing priority levels, which

require efficient management at both the data and control plane levels [48]. Hence, improved control mechanisms are still needed to manage overheads in SDN settings, e.g., controller response times and flow rule processing delays. Specifically, these solutions must take into account the queueing mechanisms in both control and data planes.

Meanwhile, other studies have also looked at modeling the behaviors of forwarding devices with SDN controllers using analytical models, e.g., such as Miao et al. [50] and Azodolmolky et al. [51]. However, in order to simplify analytical tractability, these methods have assumed basic first come first serve (FCFS) queueing and processing of all data plane flows. As a result, these solutions impose key limitations on both data and control plane packet flows, i.e., since all data flow packets are treated with the same priority level, and control plane traffic is also intermixed. As a result, this simplification can result in QoS degradation for stringent emergency response services.

Overall, the above studies assume that network control logic/intelligence is only deployed at the centralized control plane level, i.e., relegating little/no intelligence at the switching devices. As a result, the data plane handles all incoming packets in a FCFS manner regardless of their priority level. Hence these forwarding methods do not leverage any available SDN control feedback features to manage data plane queues. Now other studies have also tried to characterize incoming flows by port numbers, albeit with lower accuracy [18]. However, none of these studies have presented a practical SDN prototype that leverages intelligent control capabilities along with priority-based queueing methods (at both the data

and control planes) to minimize control response times.

2.4 Saturation and Flooding Threats in SDN

As noted earlier, SDN is being widely deployed to enhance network QoS support [28]. However recent studies have also revealed some critical, inherent security vulnerabilities in existing SDN setups [52]. As a result, researchers have also looked at improving the security of the SDN control plane. In particular, Mahout [53] proposed a mechanism to prevent flooding threats in SDN setups. However this solution was premised upon statistic aggregation techniques, which renders it impractical against data-to-control layer saturation attacks (that may exploit micro-flows). Meanwhile KernelDetect [28] presented a lightweight kernel space IDPS approach using modular string search and filtering schemes to detect and mitigate DoS threats. Also, the Avant-Guard scheme [54] detailed a solution to alleviate saturation attacks by using flow alteration techniques in OpenFlow switching devices.

Recently, Tian et al. [55] also presented FlowSec and Blackbox, two strategies to counter DoS threats by limiting the number of packets sent to the controller in a given time interval. These methods also keep track of attack levels while trying to mitigate floods. Meanwhile PrioGuard [56] detailed a DoS mitigation scheme using a non-cooperative repeated game, whereas FloodDefender [57] introduced a protocol-independent solution for DoS mitigation. Namely, FloodDefender tries to secure both the control and data layers by leveraging table-miss engineering and packet filter approaches. Meanwhile, [58] presented another solution that tries to prevent reforwarded requests-based flooding attacks in multi-

domain SDN environments. Specifically, this approach uses an adaptive rate adjustment method to change the re-forwarding rate. Additionally, MinDoS [59] used a priority manager to prevent DoS attacks. Namely, flows are forwarded to multiple buffer queues with varying priority levels to enhance controller security. Bharathi et al. in [60] also proposed a path randomization and flow aggregation-based solution to resolve the impact of DoS attacks on switch flow tables. Furthermore, SDNManager tabled a lightweight framework for DoS prevention in SDN infrastructures using flow bandwidth change forecasts [61]. Finally, [62] and [63] presented secure controller mechanisms to prevent DoS attacks against both the control and date planes. Specifically, [63] is an improved prototype version of [62] which deploys new triggers to detect and prevent DoS attacks. The proposed solution was implemented and evaluated over a hardware SDN testbed using the RYU controller software.

Some studies have also focused on SYN flooding attacks. For example, SYN flooding traffic was identified in [64] by assessing the ratio of TCP SYN packets to TCP ACK flows produced by the same entities. However, it is impractical to leverage this strategy since identifying malicious packets in large traffic pools and then accurately marking threatening flows in real-time is very challenging. Meanwhile, Deng, et al. [65] also studied SYN flooding threats in order to improve SDN resiliency. Other notable studies on SYN flooding attacks also include SLICOTS [23] and OPERETTA [24], as presented by R. Mohammadi et al. and S. Fichera et al, respectively. These IDPS schemes thwart such attacks by surveiling failed TCP SYN requests and blocking malicious adversarial hosts.

Overall, the majority of the above schemes have focused on improving the internal interactions between modules in the application-controller-switch infrastructure, i.e., to address external DoS saturation threats in SDN environments. Nevertheless, few efforts have considered the use of dynamic/self-adapting detection threshold mechanisms to handle realistic scenarios where flooding traffic and bursts may vary in real-time (exhibit time-varying patterns). Furthermore, current detection and mitigation knowledge on sophisticated SYN flooding attacks is also incomplete, and most SDN modules can themselves become vulnerable to malicious TCP SYN flood behaviors.

2.5 Open Challenges

In summary, the maintenance of QoS is a crucial concern as it dominates both infrastructure and operational costs in SDN networks (as well as legacy networks). Indeed, metrics such as controller response times, flow rule processing rates, forwarding delays, and resiliency are mutually dependent upon each other and will clearly impact overall end-to-end communication delay and achievable QoS. In this context, it is vital to develop sophisticated processing and forwarding strategies (at both the control and data plane levels) in order to maintain QoS and lower cost without affecting SDN controller overheads.

Now a typical solution here may be to dedicate more resources at the SDN controller in order to reduce its response time. However, such an approach yields significantly larger infrastructural complexity and operational costs, e.g., physical hardware and energy usage. Moreover, the processing resources available for logical controllers at a Service Provider

(SP) can still be limited, i.e., since operators may restrict the number of concurrent virtual machine instances per account. As a result, applications-specific QoS may still degrade. Furthermore, many earlier QoS studies have used port numbers to identify traffic. However, these schemes suffer from low accuracy or assume that the network can effectively classify incoming traffic. As such, practical applicability is low and most evaluation efforts have only been done in simulation-based environments. Indeed, few studies have combined and integrated priority-based flow forwarding and queueing mechanisms along with TE methods to improve response time and end-to-end traffic delay (especially in ERS networks).

Furthermore, centralized SDN controller themselves (as well as SDN-enabled switching devices) can become potential targets of saturation attacks, e.g., such as SYN flooding attacks. These DoS/DDoS attacks can further degrade the perceived QoS of ICT-based services and applications. For example, an adversary might try to exhaust resources at the SDN controller, consume excessive control plane and/or data plane bandwidth, and even overload switch flow tables [65]. Along these lines, earlier studies have proposed various solutions for addressing such threats, i.e., including, but not limited to [66], [57], [56], and [21]. However, none of these studies have proposed more effective strategies using adaptive or dynamic threshold-based kernel-level IDPS methods. Additionally, most widely-used IDPS solutions, e.g., such as Zeek [67] (formerly known as Bro) and Snort [68], yield significant system overheads and very high mitigation times (particularly Zeek).

Chapter 3 : End-to-End Latency Management in SDN Infrastructures [1]

As surveyed earlier, the centralized control capability in SDN presents a unique opportunity for providing QoS support. In particular, delay-sensitive traffic flows require effective QoS mechanisms in order to minimize user latency and controller response time. Now a key challenge here is how to handle short-term network state fluctuations (in terms of congestion and latency) while still guaranteeing end-to-end latency performance. It is here that effective path computation strategies can be developed and applied.

Along these lines, this chapter tables a systematic framework that utilizes active link latency measurements to perform efficient statistical estimation of network state and fast/adaptive path computation. The solution is also implemented in a real-world SDN controller application and tested experimentally in the Global Environment for Network Innovations (GENI), a live distributed testbed network funded by the National Science Foundation (NSF). Overall, detailed performance evaluation results shows that the proposed framework can effectively resolve end-to-end routes with minimal latency and also deliver significant reduction in controller overheads.

[1]Parts of this chapter were published by the author of this dissertation study in M. Rahouti et al. "LatencySmasher: A Software-Defined Networking-Based Framework for End-to-End Latency Optimization", 44th IEEE Conference on Local Computer Networks (LCN), 2019. The IEEE does not require individuals working on a thesis to obtain a formal reuse license (if they are the senior authors of the published work)

3.1 Problem Scope

This study focuses on end-user devices sending and receiving fixed rate real-time traffic flows. The SDN network setup uses OpenFlow switches [6] that provide access to the data forwarding plane as well as controller-switch communication (to enable state update functions). Namely, OpenFlow Version 1.5.1 is used here, although the proposed framework is readily extensible to more recent versions as well. Leveraging this setup, this chapter studies end-to-end packet delay across SDN networks using statistics collection and topological path updates. Now effective path selection implementations should try to minimize the average delay or maximum delay, or both. Efficient use of link capacity is also another key concern here [12]. Finally, the overheads associated with statistics collection and path updates can also have a significant impact on end-to-end packet processing latency at switches. Hence it is also important to address controller-switch communications overheads by reducing statistics collection overheads and lowering the amount of flow rules generated by the controller.

In light of the above, a novel framework is now proposed, leveraging centralized SDN control and topological views to manage end-to-end latency, i.e., including delays associated with path search and selection and controller-switch communication. In particular, the following techniques/methods are applied here:

- **Traffic engineering**: Combine empirical latency measurements with a heuristic path computation approach to reduce end-to-end delays

- **Statistics and flow rules offloading**: Define strategies to reduce the impact of control channel overheads on latency, i.e., speedup flow rule processing and improve statistics collection at the controller

Accordingly the research problem is divided into three different sub-problems, including statistics overhead and collection, latency metric estimation, and efficient path selection. Consider the details next.

3.1.1 Statistics Overhead and Collection

Statistics collection plays a vital role in providing real-time information to SDN provisioning applications. Now a major challenge here is reducing the impact and overheads associated with topology statistics collection/inquiries. Hence a metrics matrix (MM) is typically defined to track various statistics measurements in a given time interval, I. Specifically, these metrics can include per-switch latency, receive speed (Rx), transmit speed (Tx), hop counts, etc. This matrix can then be used to estimate link cost to further solve the path selection problem detailed subsequently (Section 3.1.3). Overall the MM matrix represents the SDN topology as a weighted graph $G = (V, E)$, where V is a set of graph vertices, each representing a switching device, and E is a set of physical links (client-switch and switch-switch). For each link $e \in E$, the vector $L(e)$ is also used to represent the current per-link metrics of a path connecting a pair of nodes. Moreover, the matrix MM can also include a respective weight function for each link $e \in E$, i.e., denoted by $w(e)$.

3.1.2 Latency Metric Estimation

Since SDN controllers must provide a high degree of globalized visibility, extracting, filtering, and deploying link and device specific statistics is of key interest. Namely, the broader objective here is to use these statistics to perform latency metric estimation and assist with effective path computation. Now in a typical communication scenario between two endpoint nodes, this model should ideally estimate the latency metrics associated with all links along the respective path. These estimates can then be used for selecting the forwarding path for network flows over a fixed time interval, I. Specifically, given the dynamicity of network topologies, any estimated values will only be considered valid for a fixed interval.

Overall, effective estimation techniques can be used to help reduce the overhead of statistics collection (via the SDN REST API). In turn, this approach can lower management plane complexity as well as per-switch latency. For example, in order to measure and collect a latency metric with OpenFlow switches, the SDN controller must repetitively transmit multiple Link Layer Discovery Protocol (LLDP) packets over a short time interval, increasing control messaging overheads.

3.1.3 Efficient Path Selection

Path selection at the SDN controller plays a critical role in ensuring end-to-end QoS. Therefore in the context of delay-sensitive user traffic, the proposed method must effectively compute a route between the communicating endpoint nodes in order to satisfy latency requirements. However, since networking topologies and link loads are highly dynamic, a

static minimum cost path will not necessarily yield the most effective route. Therefore, the proposed path selection mechanism herein jointly incorporates multiple dynamic link metrics, e.g., such as transmit/receive speeds and transmit/receive utilization per internal link, and is detailed subsequently (Section 3.2.3).

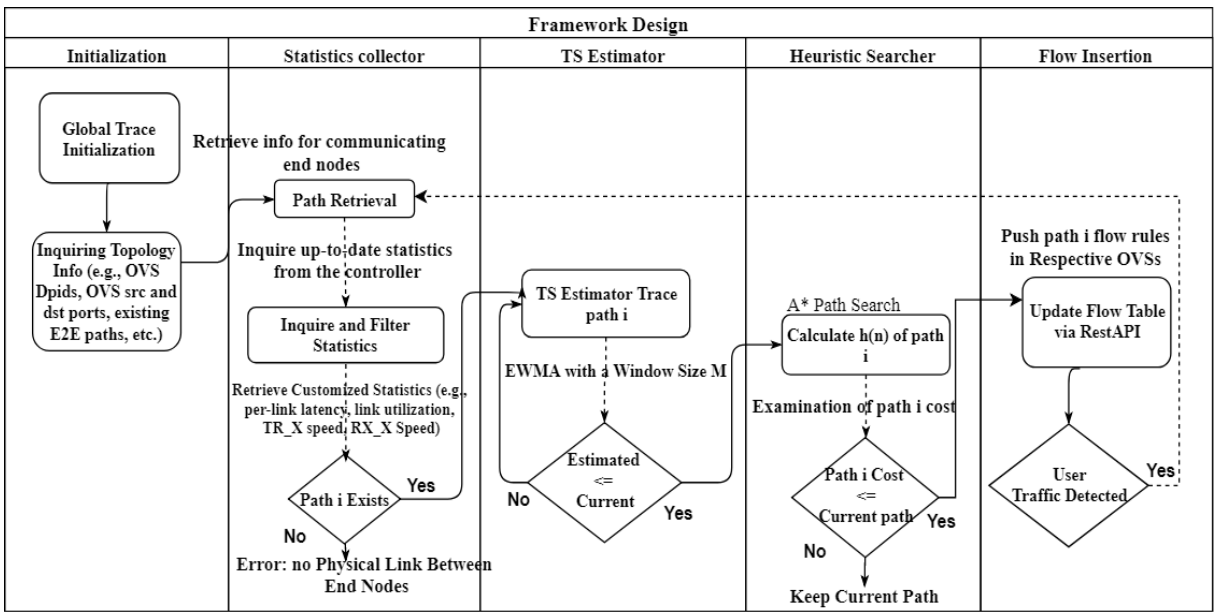

Figure 3.1: Overview of proposed framework design

3.2 Proposed Prototype

A novel solution is now presented for improving end-to-end latency for stringent real-time services. This framework is specifically designed to handle dynamic networking environments experiencing link congestion (which can cause excessive network delays, packet unsequencing, and even loss). The proposed framework is shown in Figure 3.1 and consists of three modules which are now described further, i.e., statistics collection, time series estimation, and path selection.

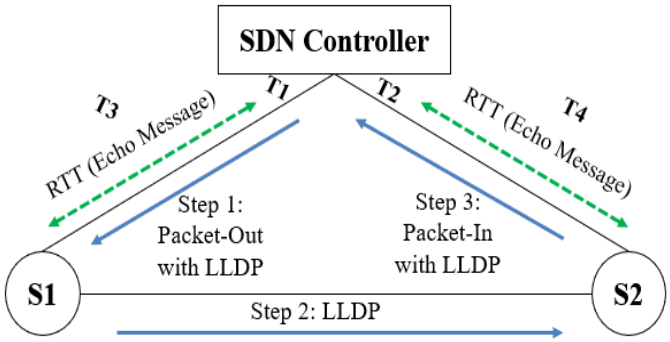

Figure 3.2: Latency metric estimation by the SDN controller

3.2.1 Statistics Collection and Latency Timing

A core component of the solution focuses on collecting link delay statistics for the SDN controller. Namely, the most recent version of the Floodlight SDN controller (Version 1.2) [32] is used to compute link latency by injecting timestamps into LLDP packets and transmitting them as *Packet_Outs* messages to each switching device, see Figure 3.2. In turn, when the SDN controller receives one of its own LLDP packets back from a neighboring switch, it processes it as a *Packet_In* and examines its timestamp value. The "elapsed time" can then be derived by subtracting the timestamp from the current time. Based upon this, the latency, Lat_L, of a link is defined as the elapsed time minus the control plane latency of the origin switch and the SDN switch (that originally sent out the *Packet_In* message):

$$Lat_L = [C_t - LLDP_t] - Lat_{SO} - Lat_{SS}, \tag{3.1}$$

where C_t, $LLDP_t$, Lat_{SO}, and Lat_{SS} represent the current time, the timestamp in the LLDP packet, the control plane latency of the origin switch, and the control plane latency of the switch that sent out the *Packet_In* message, respectively. Note that this is an approximate

calculation, since it is difficult to estimate the exact control plane latency at each switch. Now this is of particular concern in data center environments where latencies are in the sub-milliseconds range. Namely, most SDN controller clock implementations have millisecond precision, which will limit overall estimation accuracy. For instance, if a latency is manually set at a particular internal link to 10 milliseconds, the impact of this latency on the link should be feasibly observed. Hence, the LLDP-based latency should be computed as follows:

$$t\_lldp\_tx - t\_lldp\_rx - lat\_tx\_ovs - lat\_rx\_ovs, \tag{3.2}$$

where t_lldp_tx, t_lldp_rx, lat_tx_ovs, and lat_rx_ovs represent the initial timestamp values of the LLDP packet, the trip timestamp, the latency at the neighboring switch (that receives and sends out the LLDP packet to the next switch), and the latency in the next hop switch, respectively. Note that the only constraint here is for the networking environment to be stable prior to capturing the per-link latency values (and this usually occurs after about 60 seconds of wait time).

In general, most default SDN controller implementations do not provide any type of QoS support. This deficiency will clearly impact the delay performance of network applications and services with Round Trip Time (RTT) sensitivity [12] [26]. In particular, RTT instability can be affected by various factors, such as (1) network congestion, (2) in/outbound latency at the SDN switches, and (3) networking topology dynamics. To address these concerns, the proposed framework leverages the global view at the SDN controller and its computation capabilities to estimate per-link latencies. Consider the details.

3.2.2 Time Series-Based Latency Estimation

End-to-end flow latency along network paths can vary randomly as it is susceptible to dynamic changes, e.g., such as fluctuating traffic volumes, link faults, routing changes, etc. As a result, network statistics will generally exhibit a highly dependant nature, obviating the applicability of linear regression models with independent statistics assumptions. Moreover, in addition to descending or increasing trends, TS data can also show seasonality trends (whose variations will apply within a specific time window). For instance, once samples have been collected along a path for a given time window, further estimates can be derived for periods of bursty traffic. In light of the above, the proposed TS-based latency estimation treats the measured per-link latency statistics as a time series observation of a random process during time X_t. Namely, X_t is considered as a time-dependent random variable on which a realization is made, where t denotes the sample index.

Due to its holistic and global perspective, the SDN controller can obtain access to a broad range of topology statistics including, but not limited to, inter-switch statistics and both internal and external per-link measurements. The latter measurements can include link speed, transmission speed, reception speed, overall link utilization, etc. However, querying and collecting statistics measurements via the REST API and statistics collection module (at the controller) entails significant overheads at both the SDN controller and switches. As a result, the end-to-end latency of network packet traffic can be unduly affected here. Hence in order to reduce the amount of statistics inquiries made through the REST API,

the proposed TS model performs link metric estimation. In particular, per-link latency is estimated for a fixed window size based upon current topological statistics. Accordingly, a *Weighted Moving Average* (WMA) TS-based approach is now presented.

First, consider N successive observation time instances for the per-link latencies (along a particular end-to-end path) given by $z(t) = (z(1), z(2), ..., z(N)) = (z_1, z_2, ..., z_N)$. These values can be representative of N random variables distributed according to N different probability density functions, i.e., $f_1(z_1), f_2(z_2), ..., f_N(z_N)$. Hence the sequence of latency metric samples received by the SDN controller can be referred to as sample realization of per-link latencies. Now Ribeiro et al. [69] proposed a moving average technique for network bandwidth estimation by averaging recent measurements within a window size of M. Inspired by this previous work, *an Exponentially Weighted Moving Average* (EWMA) model is incorporated into the latency estimator module. Namely, this approach utilizes the per-link latency values computed by the SDN controller as described above, and generates a new estimate, Y_i, as follows:

$$Y_i = \theta_i Y_{i-1} + (1 - \theta_i) Z_i. \tag{3.3}$$

where θ_i is an exponential weighting factor, $0 \leq \theta_i \leq 1$. Note that this approach has not been proposed and implemented before for per-link latency estimation in networking-enabled environments.

Now a key requirement here is the initialization of the exponential weighting factor, θ. Namely, if θ is chosen too large, then the previous estimates will be given increased importance, and the end-to-end path latency estimation will not reflect current network changes and dynamicity. However, if θ is set to a smaller value then improved estimation agility can be achieved [70]. Ideally, however, this weighting should be adaptive such that a value θ_i is defined for each current measurement interval, i. Accordingly, a dynamic exponential weighting factor is computed as follows:

$$\theta_i = \frac{\gamma\Lambda}{\sum_{t=i-M}^{i}(Z_t - Z_{t-1})}, \quad \text{where } \Lambda = \mid Z_{max} - Z_{min} \mid, \tag{3.4}$$

i.e., Λ is the difference between the maximum and minimum per-link latency samples within the window size M. Overall, the estimation module starts with initial M and γ values of 10 and 15, respectively. In particular, these values are empirically chosen by taking into account both network topology dynamics and statistics collection overheads. Note that per-link latency samples can sometimes remain unchanged for a given duration. Hence Equation 3.4 can lead to a divide by zero error. In order to resolve this concern, the dynamic weighting factor θ_i is initialized to 0.5 if the observed per-link latencies remain constant within a given time frame.

3.2.3 Adaptive Heuristic-Path Selection

The $A*$ algorithm is widely used for graph traversal and path selection [71] due to its accuracy and performance. In particular, this scheme is an instance of the best-first search which defines a heuristic evaluation function, $f(x)$, as follows:

$$f(n) = g(n) + h(n), \tag{3.5}$$

where n is the next node on the path, $h(n)$ is the heuristic function to predict the path with the lowest cost from n to the target node, and $g(n)$ is the actual cost of the path from the initial source to the current node n, respectively. Here the heuristic function can be used to control the behavior of the $A*$ algorithm. Namely, if $h(n)$ is 0, then the actual cost, $g(n)$, drives path selection according to the ubiquitous Dijkstra algorithm for shortest path selection. Alternatively, if $h(n) = g(n)$, then the heuristic estimation is equal to the actual cost of traversing from node n to the destination. In this case the $A*$ algorithm quickly follows the lowest-cost path and does not explore other options. Finally, if $h(n) > g(n)$, then the scheme cannot guarantee that the lowest-cost path will be found, i.e., even though it runs faster since it will not expand all nodes. Hence for effective operation, the condition $h(n) \leq g(n)$ should be satisfied, i.e., meaning that the total path latency should be less than or equal to the actual cost of moving from the current node n to the target node.

Leveraging the above, the path selection module calculates the path based upon the output of the latency estimation module as follows:

$$\begin{aligned} f(n) &= g(n) + \theta_i Y_{i-1} + (1-\theta_i) Z_i, \\ i.e., h(n) &= \theta_i Y_{i-1} + (1-\theta_i) Z_i. \end{aligned} \tag{3.6}$$

As per the above, the path selection module chooses $h(n)$ to achieve dynamic path selection to the target node. Now readily-available per-link latency estimates from the latency estimation module (Section 3.2.2) can be leveraged to specify $h(n)$. Specifically, $h(n)$ is basically expressed as a set of links between two communicating nodes as follows:

$$h(< n_0, ..., n_k >) = h(n_k). \tag{3.7}$$

In order to satisfy the desired operating condition of the $A*$ algorithm, i.e., $h(n) \leq g(n)$, the following expression should hold:

$$\theta_i Y_{i-1} + (1-\theta_i) Z_i \leq g(n). \tag{3.8}$$

Overall, the above solution is capable of dynamically varying its performance according to the calculated heuristic, $h(n)$. Moreover, the shortest path between the designated end nodes is not necessarily chosen, i.e., it may yield a path that is acceptable and close to the shortest path depending upon the dynamics of a networking topology. This scheme also provides a trade-off between the speed and efficiency of path selection.

Overall, the complete implementation-level description of the proposed $A*$ scheme is also given in Figure 3.3. The destination (goal) and source nodes here are denoted by

$node_{goal}$ and $node_{start}$, respectively. Furthermore, two lists, $OPEN$ and $CLOSED$, are also maintained for tracking purposes. Specifically, $OPEN$ is the list of pending tasks, i.e., nodes that are visited but not expanded (and their successors are not searched yet). Meanwhile, $CLOSED$ is a list of nodes that have already been visited and expanded, i.e., where the successors have been explored and placed in the $OPEN$ list.

```
1: Initialize OPEN list & TS − estimator with M = 10
2: INPUT: G(V, E), c^{ij} ∀(i, j) ∈ E, R, F, D
3: OUTPUT: path vector
4: while the OPEN list is != empty
5:    f(node_c) = g(node_c) + h(TS − estimator(node_c))
6:    if node_c is node_g
7:       return path & break
8:    Generate each state node_s that comes after node_c
9:    for each node_s of node_c
10:      Set successor_cost = g(node_c) + h(TS − estimator(node_c, node_s))
11:      if node_s is in the OPEN list
12:         if g(node_s) ≤ successor_cost
13:            Go to line 20
14:      else if node_s is in the CLOSED list
15:         if g(node_s) ≤ successor_cost
16:            Go to line 20
17:         Move node_s from the CLOSED list to the OPEN list
18:      else
19:         Add node_s to the OPEN list
20:         Set h(node_s) to be the heuristic distance to node_g
21:      Set g(node_s) = successor_cost
22:      Set the parent of node_s to node_c
23:   Add node_c to the CLOSED list
24: return path vector
```

Figure 3.3: Adaptive $A*$ path selection algorithm

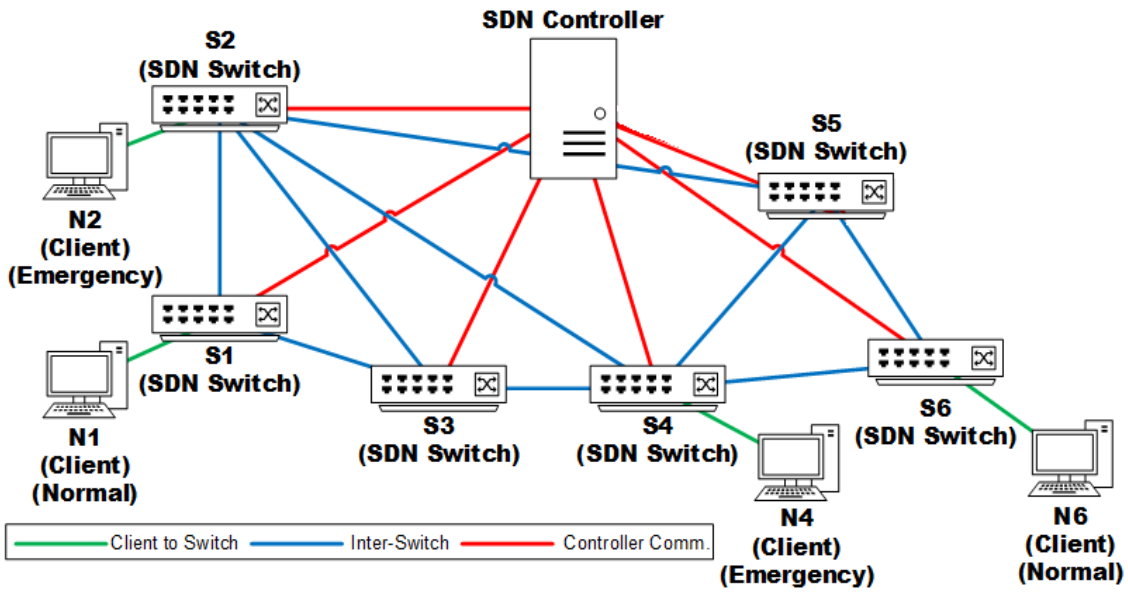

Figure 3.4: The experimental SDN topology on NSF GENI testbed

3.3 Performance Evaluation

The proposed latency management solution is implemented and evaluated in a live SDN network testbed. In order to gauge the proposed framework and conduct a comprehensive empirical examination of end-to-end latency, three key requirements constraints are incorporated when building the SDN topology. Foremost, open source SDN controller and OpenFlow software implementations are chosen to ensure wider adoption/evaluation of the work. Next, realistic network traffic emulation is done using a dedicated software module. Finally, the network topology is designed with multiple paths and physical links with changeable loss rates and link speeds. As noted earlier, the solution is implemented in the NSF GENI [72] network, a real-world federated and heterogeneous facility. This is a well-

established testbed that allows researchers to build arbitrary network topology slices and deploy tailored end host (traffic generation) and SDN controllers.

Now in order to comprehensively evaluate end-to-end latency performance over realistic networks, the GENI topology is chosen to interconnect nodes from Lexington, Kentucky, Cleveland, and Louisiana, as shown in Figure 3.4. Namely, the stitching capability in the GENI portal allows researchers to interconnect nodes from multiple aggregates (e.g., cities) to build a larger coherent topology. Furthermore, each switching node also runs the Open vSwitch (Version 1.5.1) protocol, whereas the SDN controller host deploys Floodlight (Version 1.2), an open source SDN controller solution.

3.3.1 Per-Link Latency Examination and Path Selection

Initial tests are done to evaluate the relative percentage error of the per-link latency estimates. Namely, the difference between estimated calculations based upon the TS model and the actual measured values (calculated by the SDN controller) is derived as follows:

$$\begin{aligned} Error &= \frac{|Latency_E - Latency_A|}{Latency_A}, \\ \text{i.e., } Error &= \frac{|[\theta_i Y_{i-1} + (1-\theta_i) Z_i] - Latency_A|}{Latency_A}, \end{aligned} \tag{3.9}$$

where $Latency_E$ and $Latency_A$ represent the estimated latency and the actual measured value, respectively. Specifically, the relative error is calculated as the absolute error divided by the magnitude of actual latency values (expressed in terms of percentage) as per Equation 3.9 for different observation trials. Based upon the above, Figure 3.5 plots the average relative percentage error of multiple links in a single end-to-end path. Namely, the given path lies

between end nodes N_1 and N_6, and uses Link 1, Link 2, and Link 3 (representing inter-switch links $S_1 \rightarrow S_3$, $S_3 \rightarrow S_4$, and $S_4 \rightarrow S_6$, respectively), see figure Figure 3.4. As expected, in the initial steps the estimated latencies will equal the actual values since the TS estimator uses the initially-observed per-link latency values to compute the averages (and these will closely match the actual latencies measured). However, with increasing measurement samples, the accuracy of path latency estimation improves notably, and the average error rate remains below 0.2% as shown in Figure 3.5.

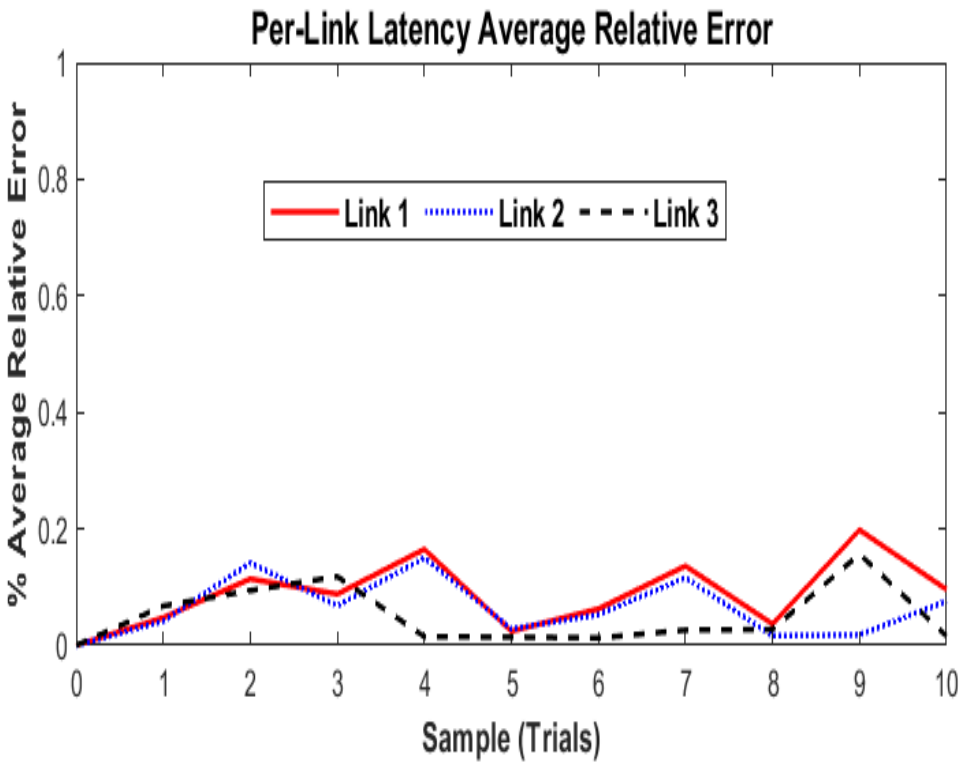

Figure 3.5: Average relative error of individual links in a single end-to-end path

Next, the actual and estimated path costs are compared to evaluate the effectiveness of the proposed adaptive $A*$ path selection algorithm. As discussed in Section 3.2, the

$A*$ algorithm will yield an improved path if the heuristic condition is satisfied in Equation 3.8. Hence Figure 3.6 plots both the actual and estimated latency values based upon the TS estimator module. These results confirm that the the actual path cost is only smaller than the estimated cost in the initial TS estimator phase, i.e., since the exponential moving average of collected latency values is likely higher than the actual path cost.

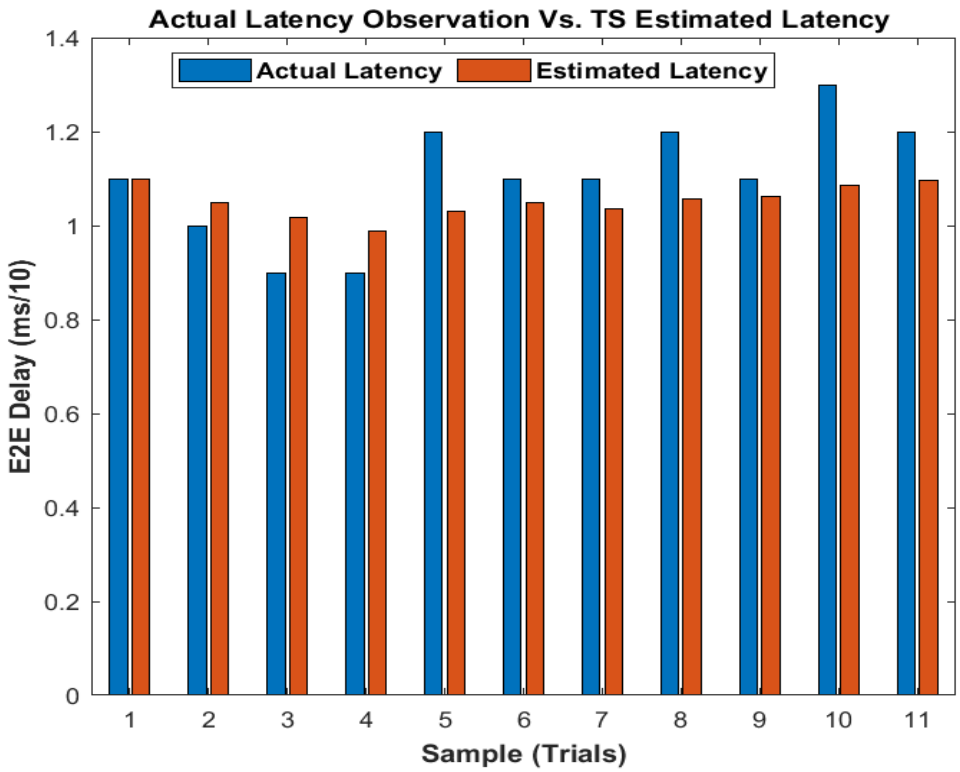

Figure 3.6: Path selection comparison (actual versus estimated path cost)

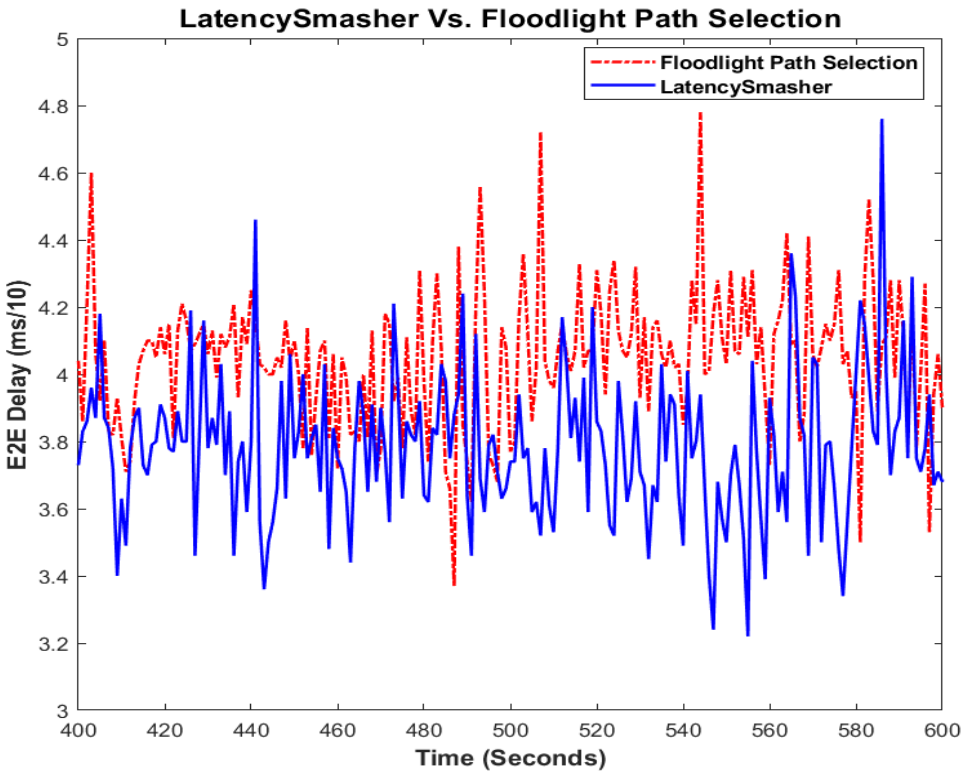

Figure 3.7: Latency comparison between $A*$ approach and default Floodlight path selection

3.3.2 Proposed Framework Performance Versus Default Path Computation

The proposed adaptive $A*$ scheme is also compared to the default path selection algorithm in the Floodlight SDN controller. Namely, Figure 3.7 plots the throughout comparison between these two methods. Specifically, in this experimental setup, N_1 and N_6 are designated as the source and target nodes, respectively. Overall, these results confirm that the proposed framework outperforms the Floodlight path selection mechanism with regard to end-to-end latency minimization. Namely, as long as the heuristic selection condition holds (as discussed in Section 3.2.3), the adaptive $A*$ algorithm only expands on nodes with

lower heuristic latency values. This approach tries to reach the destination node as quickly as possible rather than expanding every other node like the Dijkstra algorithm. Moreover, the modified path selection module is only invoked when a better (lower) cost estimate for the current path is found by the TS estimator module.

Table 3.1: Percentage improvement in latency (proposed framework versus default controller)

Measurement Statistics	**Node N_1**	**Node N_2**	**Node N_3**
Mean	3.07 %	6.41 %	4.51 %
Median	3.82 %	5.95 %	5.07 %

Additionally, Table 3.1 also summarizes the percentage improvements in end-to-end latency yielded by the proposed framework (versus the default Dijkstra path selection method provided in the Floodlight controller). These results confirm that the proposed SDN-enabled framework reduces delays by at least 3% (for the different nodes communicating with the designated server node, N_6).

3.3.3 Overhead Considerations

Finally, control overheads are also measured to gauge operational complexity. Specifically, the SDN controller CPU resource utilization is plotted for both the proposed framework and default Floodlight controller in Figure 3.8. Again, these results demonstrate that the developed framework outperforms the existing Floodlight controller implementation, i.e., average overhead reduction of about 20%. Overall, this improvement is mainly due to the increased statistics collection tasks running at the Floodlight SDN controller. Specifically, the default statistics collection feature generates a large amount of unnecessary topology and

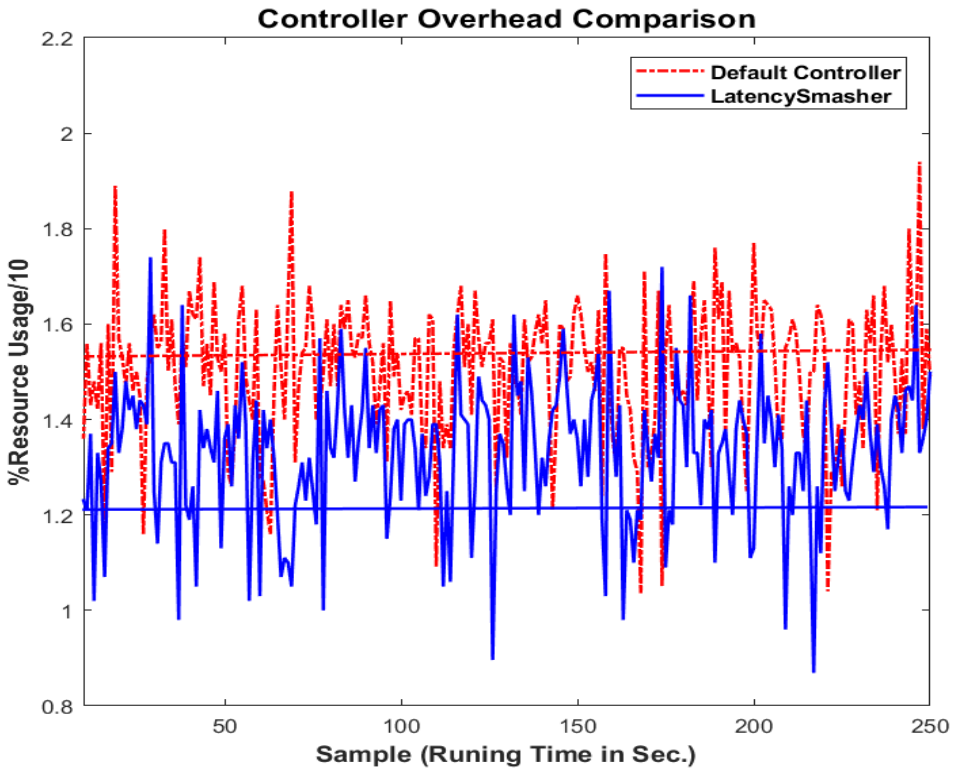

Figure 3.8: Overhead comparison with default Floodlight (horizontal lines plot averages)

per-link statistics in a periodic manner (every 10 seconds by default). These excessive calculations and exchanges result in notably higher controller and switch overheads. By contrast, the proposed framework only requests individual per-link statistics when networking traffic is initiated by an end node. Similarly new flow rules are only pushed to switches when an alternative path with a better cost is estimated. Again, this contrasts with the default path computation facility in the Floodlight controller, i.e., as the proposed adaptive path search only expands along a new path if its heuristic cost (from TS estimation module) is less than the current cost.

Chapter 4 : SDN-Based Priority Queueing

As surveyed earlier, SDN technology provides separation between the data and control layers in a network in order to enhance service provisioning and management. Now for delay-sensitive client services, e.g., such as an emergency response, various QoS strategies can be implemented using SDN, e.g., such as minimization of end-to-end delay, effective calculation of forwarding paths, and minimization of SDN controller response times. Furthermore, priority-based traffic management can also be introduced for both data and control plane packet flows.

Along these lines, this chapter presents a novel QoS solution that leverages the global control capabilities of SDN along with priority-based queueing and TE techniques. Termed as QoS Priority (QoSP), the proposed solution leverages multiple queues to handle higher-priority traffic while preempting control plane traffic. In particular, the proposed scheme distributes traffic across multiple queues with different priorities at each switch port (in the OpenFlow devices). Furthermore, the solution is also implemented in a live SDN controller and tested in the NSF GENI testbed facility. Overall, detailed performance evaluation results show that the proposed QoSP scheme can effectively resolve controller latency concerns and manage queueing times for each traffic priority in the data plane.

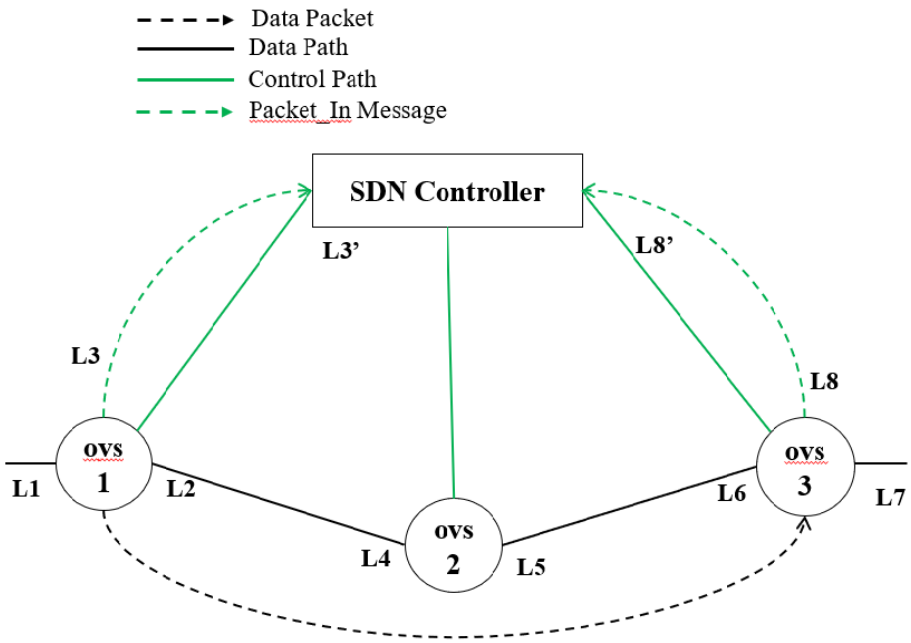

Figure 4.1: Latency overview in a SDN setup

4.1 Problem Scope

As noted earlier, OpenFlow SDN-enabled networks operate in a reactive manner by checking switch flow tables for existing rules for incoming packets. If no matching flow rules are found, switches send OpenFlow *Packet_In* messages (which include first packet headers) to the SDN controller. Subsequently, the controller replies with *Packet_Out* messages to allow the switches to install respective forwarding rules in their flow tables (assuming that valid routing paths are found between the nodes). Now clearly control and switching latencies can have a notable impact on overall packet routing delays here.

The overall breakdown of latency and messaging control is further illustrated in Figure 4.1 for a sample three switch network. Namely, the end-to-end route here traverses three Open vSwitches, i.e., OVS_1, OVS_2, and OVS_3. Accordingly, the flow enters the switches at times L_1, L_4, and L_6 and then departs at times L_2, L_5, and L_7, respectively. Now assume that flow entry matching procedures at OVS_1 and OVS_3 incur additional delays (overheads) from $Packet\_In$ messaging with the SDN controller. Hence the actual delay between OVS_3 and OVS_1 is given by L_7 - L_2. Additionally, L_3 and L_8 represent the timestamps for the $Packet\_In$ messages departing OVS_1 and OVS_3, respectively. In turn, this gives $\hat{L_3}$ and $\hat{L_8}$, the corresponding arrival times of $Packet\_In$ messages at the SDN controller. Note that d_1 and d_3 also denote the propagation delays between the SDN controller and OVS_1 and OVS_3, respectively, and these values are non-negligible in larger metro-core networks. In this context, the minimization of controller-switch communication latency is of key importance. As a result, priority-based queueing is proposed to alleviate these delays and control the scheduling management of control packets.

Now existing OpenFlow implementations only support a basic FCFS queueing strategy to handle all ingress data flows [6]. However, this limitation can prevent data plane flows from achieving their stringent QoS requirements in case of path congestion. To address this concern, reordering of incoming traffic can be done in the data plane queue, i.e., according to specific priority levels. In addition to multiple data plane queues, a control plane queue can also be incorporated into the overall control mechanism to manage operation (via feed-

back control from the SDN controller). Overall, this approach can reduce controller response times and packet processing/forwarding delays based upon different priority levels.

4.2 System Overview

The QoSP provisioning framework for SDN is now presented. Foremost, this solution classifies incoming data traffic flows (packets) into multiple priority levels and buffers them separately. A feedback control mechanism is then used to implement priority-based transmission of all enqueued data packets, i.e., with the SDN controller performing dynamic queue control.

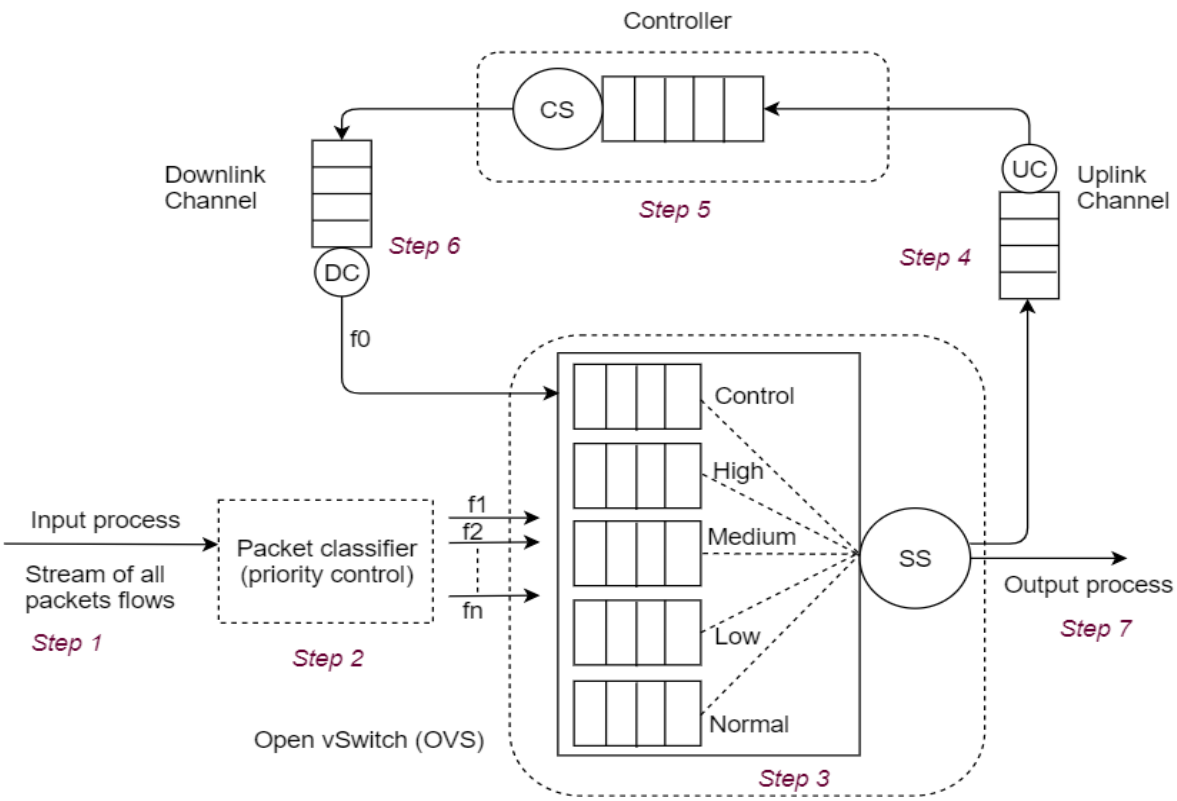

Figure 4.2: Overview of SDN-based priority queueing structure (QoSP)

The overall SDN queueing setup is shown in Figure 4.2 for both the data and control planes in the QoSP scheme. This design relies upon a set of queues. Foremost, the uplink channel (UC) and downlink channel (DC) queues buffer switch-to-control and control-to-switch control packets, respectively. These queues are also assumed to have infinite length. Meanwhile, the SDN controller maintains a finite queue to store incoming lookup requests (*Packet_In* messages). Finally, the Open vSwitch also maintains a series of priority-based queues for incoming data packets. In particular, four different priority levels are supported here, i.e., normal, low, medium, and high (Figure 4.2). Overall, the control plane queues (UC, DC, and controller) are intended for flow rules management and control, and the corresponding data plane queues store traffic with different priority levels. Furthermore, control-related packets are also prioritized over all data plane packets regardless of the priority level of the latter. Hence during flow scheduling, packets in the control channel queue are guaranteed a minimal processing latency, whereas data plane packets are processed according to their arrival priority levels.

Now consider a typical sequence of queueing actions in the above QoSP setup. Foremost, incoming data flows arriving at the Open vSwitch device (Step 1) are classified and placed in the data plane queue in an ordered manner according to their corresponding priority level. Note that packets will only be queued if the buffer is not full and the switch server (SS) is not busy (Step 2). Now assuming the SS is idle, a packet will be processed and forwarded immediately if a matching flow rule entry is found in the Open vSwitch flow

table (Step 7). Alternatively, the header of the incoming packet will be forwarded to the SDN controller (in a $Packet\_In$ message) via the UC to acquire a corresponding flow rule entry (Steps 4-6). Further analysis and details of the proposed solution are now presented, including a detailed analytical queueing model.

4.2.1 System Queueing Model

The end-to-end latency performance of the proposed QoSP solution is now modeled using queueing theory. In particular, the focus is on capturing control response and data-control channel latencies (including queueing delays). For tractability, it is assumed that the packet arrival rates and service times follow Poisson and negative exponential distributions, respectively. As a result, the uplink (UC) and downlink (DC) channels can be modeled as ubiquitous $M/M/1$ queueing systems, whereas the SDN controller can be modeled as a $M/M/1/K$ queueing system, where K denotes the maximum number of $Packet\_In$ messages that can be stored in the SDN controller. Consider the details.

End-to-end latency in SDN involves multiple factors, including propagation delay, switching latency, and control latency. In general, the processing of incoming flows (at a switching device) can involve significant overheads depending upon flow rule matching procedures. Namely, unlike packets that match with existing flow rules in the switch flow table, unmatched packets must undergo additional delay as they are forwarded to the SDN controller, i.e., table miss. Therefore, the end-to-end latency for data packets will also depend upon the control latency and response time. Hence consider a pair of communicating end

nodes, n_1 and n_2, and the flow route between them as a sequence of single hops $r_d^{1,2} =< h_{n_1} \rightarrow O_1, h_{O_1} \rightarrow O_2, ..., h_{O_i} \rightarrow n_2 >$, where h and O are the corresponding links and switches, respectively. Now first consider the end-to-end latency in case of a table lookup match. Namely, if the incoming flow matches an existing flow rule in the switch flow table, the total end-to-end latency is given by:

$$L_{1,2} = Pg^{n_1} + \sum_{i=1}^{j+1} Tr^i + \sum_{i=1}^{j} (Q^i + Pr^i), \tag{4.1}$$

where Pr^i and Q^i are the processing delay and queueing delay at the $i\text{-}th$ switch, respectively, Tr^k is the transmission latency at the $i\text{-}th$ link, and Pg^{n_1} is the propagation delay at end node n_1. However, if the arriving flow does not match any existing flow rule, additional controller latency and queueing delays are incurred. Hence in this case, the end-to-end latency, $\hat{L}_{1,2}$, is given by:

$$\hat{L}_{1,2} = Pg^{n_1} + \sum_{i=1}^{j+1} Tr^i + \sum_{i=1}^{j} (\hat{Q}^i + Pr^i) + C_{1,2}, \tag{4.2}$$

where $C_{1,2}$ and $\hat{Q}^i$ represent the additional control latency and Open vSwitch queueing delays. Hence based upon Equations (4.1) and (4.2), the control path latency, $C_{1,2}$, can be written as:

$$C_{1,2} = \hat{L}_{1,2} - L_{1,2} + \sum_{i=1}^{j} (Q^i - \hat{Q}^i). \tag{4.3}$$

In addition to the above, end-to-end latency is another critical parameter. Namely, this value, termed as L_{e2e}, can be derived in terms of the flow rule entry hit or miss proba-

bility. Specifically, the average delay experienced by an incoming packet is given by:

$$L_{e2e} = L_{hit} * P_{hit} + L_{miss} * (1 - P_{hit}), \tag{4.4}$$

where L_{hit} and L_{miss} denote the average latencies for packets with matching and non-matching (miss) flow rule entries, respectively, and P_{hit} is the probability that a matching flow rule is found at the switch. Based upon the above, L_{hit} and L_{miss} can be estimated as follows:

$$L_{hit} = \frac{L_{e2e} - L_{miss} * (1 - P_{hit})}{P_{hit}}, \tag{4.5}$$

and

$$L_{miss} = \frac{L_{e2e} - (L_{hit} * P_{hit})}{1 - P_{hit}}. \tag{4.6}$$

Extending upon this, the average queueing delay in the data plane at switch i, Q^i, can be derived from Equation 4.4 by subtracting the queueing delays at the SDN controller and UC and DC queues as follows:

$$Q^i = L_{hit} * P_{hit} + L_{miss} * (1 - P_{hit}) - (D_u + D_d + D_{sdn}), \tag{4.7}$$

where D_u, D_d, and D_{sdn} denote the uplink, downlink, and SDN controller delays. Now let D_{ls} summarize these delays as follows:

$$D_{ls} = D_u + D_d + D_{sdn} \tag{4.8}$$

Hence the average waiting time per packet in the data plane queue can be calculated by subtracting D_{ls} from the total end-to-end latency, L_{e2e}. Since control plane packets are

assigned the highest priority and served in a non-preemptive forwarding manner at the switches, the service rate for these packets will be smaller than that of data plane packets. Therefore, given an arrival rate of λ_{sdn} at the control plane queue and an average queueing time in the data plane queue, Q^i, from Equation 4.7, the waiting time for normal packets (lowest priority) is given by:

$$w_p = D_{re} + D_{ovs} + D_{oh}, \tag{4.9}$$

where D_{re}, D_{ovs}, and D_{oh} represent the residual packet service time, average switch waiting time, and additional waiting time due to higher priority control plane packets, respectively. Namely, Equation 4.9 represents the average time a low priority packet must wait prior to being served and forwarded by the switch server (assuming that the corresponding flow forwarding entry exists). Note that D_{re} can also be obtained by using the utilization law [73] as follows:

$$D_{re} = \frac{\lambda^i_{ovs}}{\mu^2_O}. \tag{4.10}$$

where μ^2_O and λ^i_{ovs} are the packet service and packet arrival rates at the switch i, respectively. Additionally, by applying Little's Law, and assuming a service rate of μ_{ovs} packets/sec at the switch, the average packet waiting time at the switch, D_{ovs}, is given by:

$$D_{ovs} = \frac{\lambda_{ovs} * (D_{re} + D_{ovs} + D_{oh})}{\mu_O}, \tag{4.11}$$

Meanwhile, the added waiting time due to control plane packets, D_{oh}, is given by:

$$D_{oh} = \frac{D_{re} + D_{ovs} + D_{oh} * \lambda_{sdn}}{\mu_O}. \tag{4.12}$$

Finally, the average packet delay in the data plane for normal packets (lowest priority) can be derived by substituting Equations 4.10, 4.11, and 4.12 into Equation 4.9. Note that both Equations 4.9 and 4.4 will be used later in Section 4.3 to measure QoSP scheme performance.

4.2.2 Queue Control Mechanism

The proposed QoSP scheme also uses the well-known Random Early Detection (RED) scheme [74] to manage all the data plane queues. Namely, this probabilistic mechanism increases packet drop probability after the average queue length crosses a specified threshold (up to the maximum buffer size). Since RED operation requires detailed queue level information at the switch, a utilization monitor is also placed at the REST API to periodically gather data plane queue statistics as well as port statistics (at OpenFlow switches). This information is then sent to the SDN controller for each queue (priority level). Specifically, data plane queues are instantiated by using an available port number at the switch. Detailed statistics for each data plane queue are then collected by the utilization monitor by matching the switch identifier with the queue instantiation port number, i.e., by using the REST API $Port_{usage}(OVS_1, Port_2)$ function.

The pseudo-code of the overall RED-based queue control mechanism is also shown in Figure 4.3. This scheme tries to control queue length and avoids dropping packets from higher priority data flows. To achieve this, the average queue size is calculated for each data plane priority queue level. In particular, these averages are updated whenever a priority level queue is idle, i.e., to keep track of how many packets can be forwarded during the idle

```
1: INPUT: Packet_In; Queue_Stats
2: OUTPUT: Feedback signal
3: for each packet p_i
4:    Calculate average queue size avg_q
5:    Maintain avg_q running average of queue length
6:    if avg_q < min_th
7:       Low queuing −− > send packets through
8:    if avg_q > max_th
9:       Drop packet
10:      Protection from misbehaving sources
11:   else
12:      Mark packet proportional to queue length
13:      Notify sources of incipient congestion
14: return Packet_Out
```

Figure 4.3: Queue control mechanism

interval. Now based upon the feedback control signal message from the SDN controller, if one or more data plane queues are full (i.e., congestion), the switch is instructed to drop packets with lower priorities (than the one arriving at the switch ingress port). Hence the probability that a given priority queue experiences packet drops increases as more flows with higher priority levels arrive at the switching device.

Finally in order to ensure that control packets (i.e., $Packet\_In$ messages) are properly prioritized over data plane packets, the solution also leverages the OpenFlow *Pause* controller feature. Specifically, this action allows the switch to pause a packet's forwarding procedure based upon its flow table (and serializes the packet state as *continuation* in the $Packet\_In$ message). Later on, the SDN controller transmits a continuation flag back to the Open vSwitch using the NXT_RESUME message, i.e., to resume processing of a data plane packet from its previous interruption point.

4.3 Performance Evaluation

The proposed QoSP traffic management solution is also implemented and evaluated in the NSF GENI testbed. In particular, the same overall topology and (hardware, software) setup configuration is used from the previous chapter, as shown in Figure 3.4. Meanwhile, traffic generation and measurement is also done using a variety of tools, including iPerf3, ping, and Linux-based traffic control via the TC package [75]. Detailed results are now presented for control latency, service fairness, and end-to-end service delay.

4.3.1 Control Latency

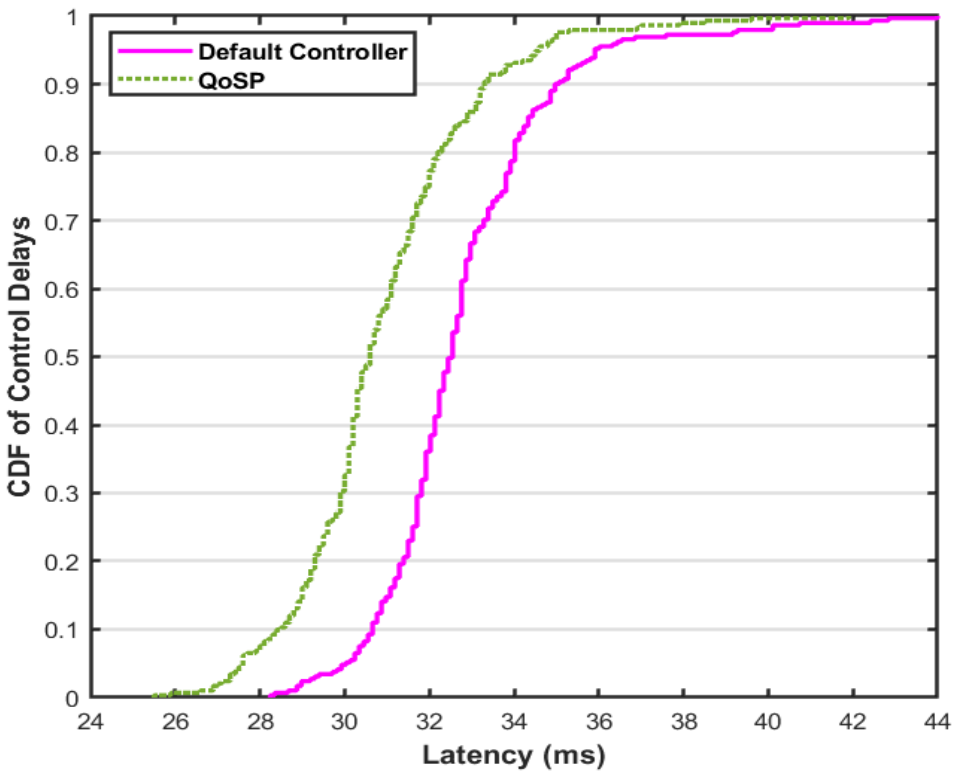

Figure 4.4: Control latency in QoSP solution versus default Floodlight controller

Control path latency is evaluated for data transfer between nodes N_1 and N_2 along the route $N_1 \rightarrow S_1 \rightarrow S_2 \rightarrow N_2$ (in Figure 3.4). Now based upon Equation 4.3, the control plane latency, $C_{1,2}$, can be approximated as:

$$C_{1,2} \approx L_{1,2} - \hat{L_{1,2}} \tag{4.13}$$

Similarly, the reverse path delay from node N_2 to node N_1, $C_{2,1}$, can also be approximated as $C_{2,1} \approx L_{2,1} - \hat{L_{2,1}}$. Hence the total control plane latency can be represented as the sum of $C_{1,2}$ and $C_{2,1}$ as follows:

$$C \approx (L_{1,2} + L_{2,1}) - (\hat{L_{1,2}} + \hat{L_{2,1}}) \tag{4.14}$$

Note that $L_{1,2} + L_{2,1}$ represents the overall RTT for packet miss instances, i.e., no flow rule entry match in the switch flow table. Meanwhile, $\hat{L_{1,2}} + \hat{L_{2,1}}$ represents the RTT of the same packet experiencing a table hit, i.e., flow rule entry found in the Open vSwitch table. Hence the control plane latency between the pair of nodes N_1 and N_2 can effectively be computed by simply subtracting packet RTT values. Accordingly, Figure 4.4 compares the control latency of the proposed SDN solution versus the un-optimized SDN controller. Here the buffer size at the SDN controller is set to 10 kilobytes, whereas the transmission rates of the UC and DC queues are set to 100 packets per second. Overall, the Cumulative Distribution Function (CDF) plots show a notable improvement/reduction in average control path delay with the QoSP scheme. In particular, the average delay is about 8% lower, i.e., mean 26 ms for QoSP versus 34 ms for the default controller.

4.3.2 Service Fairness

Fairness between different data traffic priority queues is also a key concern. For example, higher priority flows can easily degrade the throughput performance of lower priority flows resulting in bandwidth starvation for some users. Hence the service fairness of data plane queueing is also examined by using the global fairness index introduced by Jain [76]:

$$Fairness_{Index} = \frac{(\sum X_i)^2}{n \sum X_i^2}; \tag{4.15}$$

where

$$X_i = \frac{T_i}{O_i}, \tag{4.16}$$

and T_i and O_i denote the measured throughput (using iPerf3) and fair (average) throughput for sample i, respectively. Namely, this index gauges service performance based upon the measured and expected throughputs.

Accordingly, Figure 4.5 plots service fairness results for both the QoSP solution and default Floodlight implementation (FCFS queueing). In particular, the index is measured as a function of the table hit probability in the Open vSwitch flow table. Overall, the findings in Figure 4.5 show that the proposed QoSP yields much higher global service fairness. This improvement is a direct result of the selective queueing mechanism used in this solution. Specifically, the default setup does not prioritize control packets over data packets, thereby impacting both queueing and switching delays for data flows (and high priority flows in particular). By contrast, the proposed QoSP scheme yields notable improvement in service

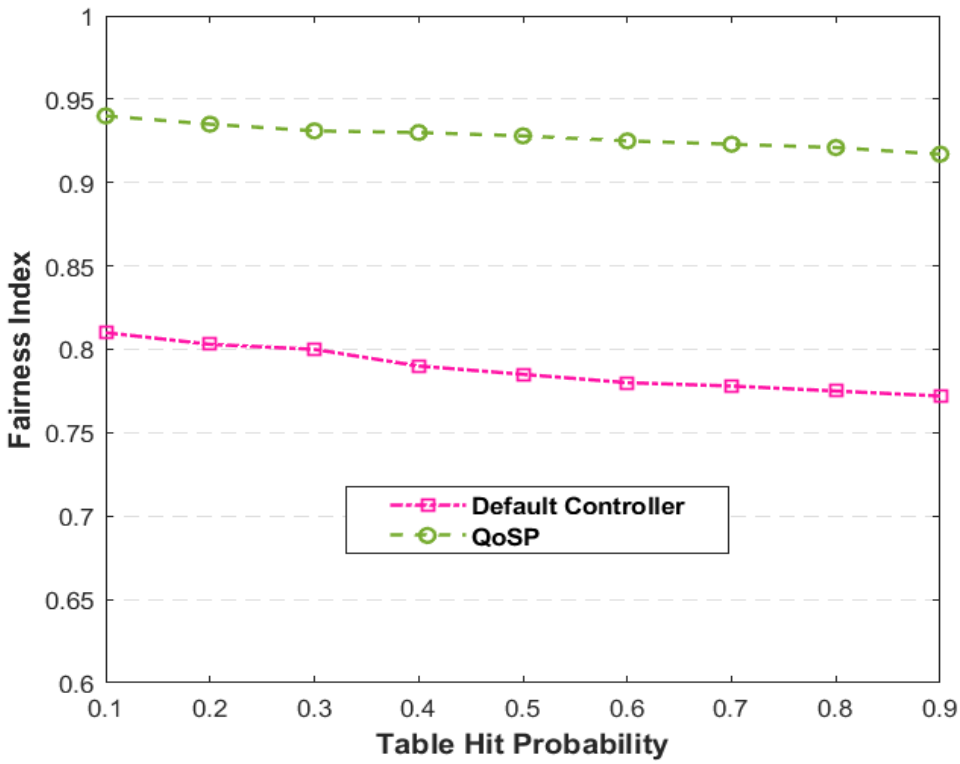

Figure 4.5: Global service fairness index comparison

fairness for both control and data plane packets with respect to table miss and table hit probabilities.

4.3.3 Priority Level Delay Variation

The prioritization of data flows is crucial for providing adequate QoS support for high-end user services, e.g., such as emergency response. Hence to further examine the performance of the proposed QoSP scheme, end-to-end latency is empirically measured for each data plane priority level, i.e., switch queueing delays. Now since the total queueing time is directly impacted by link delay, flow latency is gauged for two different link delay

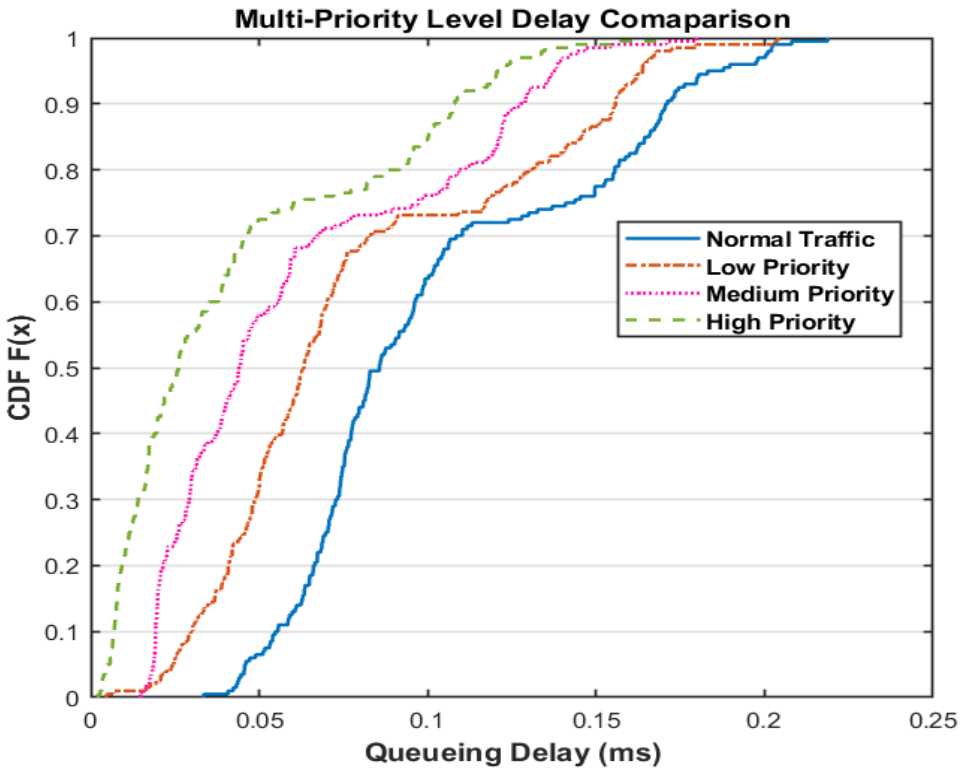

Figure 4.6: Queueing delay per priority level (normal to high for QoSP scheme)

variations between nodes N_1 and N_2 in Figure 3.4, i.e., termed as low and high. Namely, the link rates in the SDN topology are set to 5 Mb/s and 500 Mb/s for the low and high scenarios, respectively. Additionally, data traffic is also generated at a speed of 5 Mb/s and 500 Mb/s for the low and high scenarios, respectively. Subsequently, the latencies are computed for all four data plane priority levels (normal, low, medium, and high) using 1 second time intervals. Accordingly, flows with different priority levels are then transmitted from node N_1 to node N_8 in Figure 3.4, and the overall queueing delay is measured for each flow. The respective delay CDFs for each priority level are then plotted in Figure 4.6. As expected, higher-priority flows have the lowest delays. For example, almost 75% of higher

priority packets experience under 0.05 ms delays. By contrast close to 65% and 40% of medium and low priority packets, respectively, experience smaller delays. Meanwhile, very few normal traffic flows (i.e., lowest priority) have end-to-end delays under 0.05 ms, i.e., only about 10%.

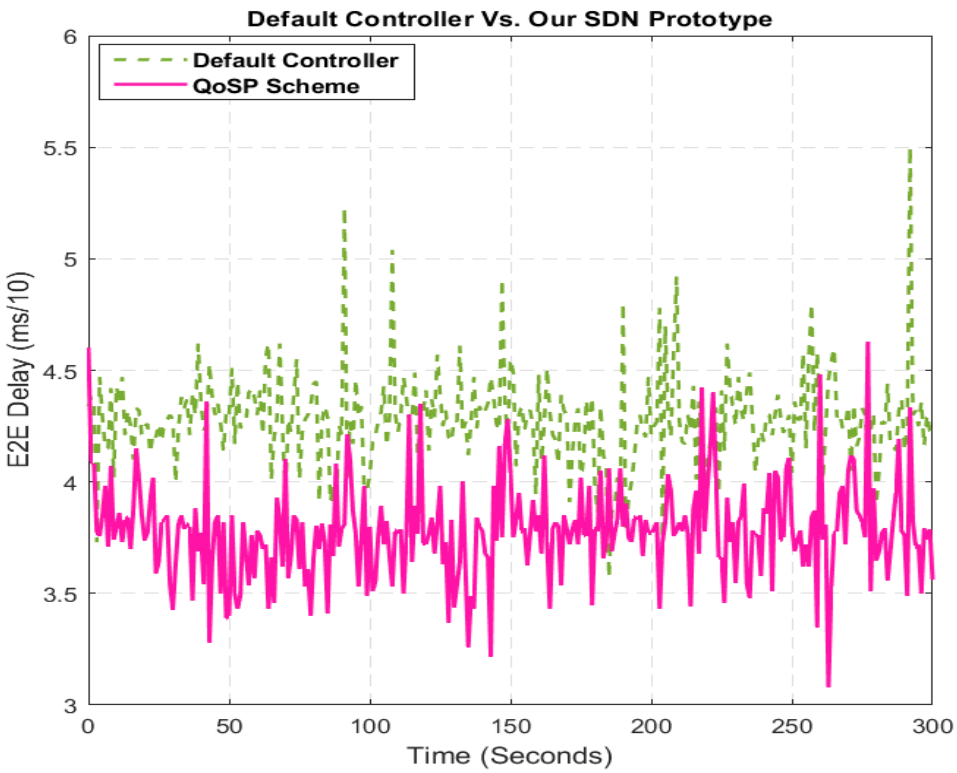

Figure 4.7: End-to-end latency comparison

4.3.4 End-to-End Delay Validation

Finally, Figure 4.7 plots the throughout comparison between the proposed SDN solution and the default SDN Floodlight controller. In this case, both nodes N_1 and N_6 (in Figure 3.4) are designated as the source and target entities, respectively. Overall, the results

demonstrate that the QoSP prototype notably outperforms the default forwarding mechanism in the Floodlight SDN controller, i.e., with respect to end-to-end latency. Namely, the results in Figure 4.7 show that the proposed solution reduces overall end-to-end latency by at least 5% for data transfer between N_1 and N_6 nodes, i.e., along the route $N_1 \rightarrow S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow S_6 \rightarrow N_6$.

Chapter 5 : Dynamic Threshold-Based SYN Flood Attack Detection

DoS and DDoS attacks, and in particular TCP SYN flooding attacks, pose serious threats to SDN-based network control setups. Hence a variety of IDPS schemes have been introduced for identifying and preventing such occurrences, as surveyed in Section 2.4. However, most of these schemes yield significant performance overheads and response times, making them inflexible and inapplicable in large-scale operational networks.

Therefore in order to address these concerns, this chapter presents a novel adaptive dynamic threshold-based kernel-level IDPS that leverages SDN capabilities to handle TCP SYN flooding attacks. The proposed solution is then evaluated to detect and mitigate the aforementioned threats and also compared against traditional IDPS technologies, namely Snort and Zeek. These tests are done using a mixture of fundamental adversary attacks, as well as SDN-specific threats in the real-world GENI testbed. Overall, detailed experimental results demonstrate the efficacy of the proposed scheme within SDN environments.

5.1 Research Background and Problem

In an SDN-enabled network, switching devices (such as Open vSwitch platforms) process packet flows according to the rules injected by the SDN controller. Namely, these flow rules are generated and transmitted by the SDN controller to data plane switches in either a proactive or reactive manner. Specifically in the proactive mode, the controller tries to populate rules prior the arrival of user traffic at the forwarding device. Conversely, in the reactive mode, the controller injects and adjusts flow rules dynamically in real-time. Hence this study focuses on using reactive techniques to handle TCP SYN flooding threats against SDN controllers.

5.1.1 Overview of SYN Flooding Attacks

TCP SYN flooding attacks are DoS attacks that try to overwhelm a victim's host computer with a large quantity of ICMP, SYN/SYN fragments, or UDP packet traffic. In particular, these attacks appear when a single or multiple hosts are flooded by TCP SYN segments initiating unaccomplished TCP connection requests (and hence are unable to respond to legitimate connection requests) [77]. Specifically, TCP client and server hosts use a $three-way$ handshake mechanism to establish a connection, as shown in Figure 5.1. Successful setup requires the client to send a SYN packet to the server host, which in turn replies with a SYN_ACK packet and immediately allocates TCP stack resources for this established connection query [78]. Once this is achieved, the server enters a listening state, called Transmission Control Block (TCB) [23], and stays there until it receives a final ACK

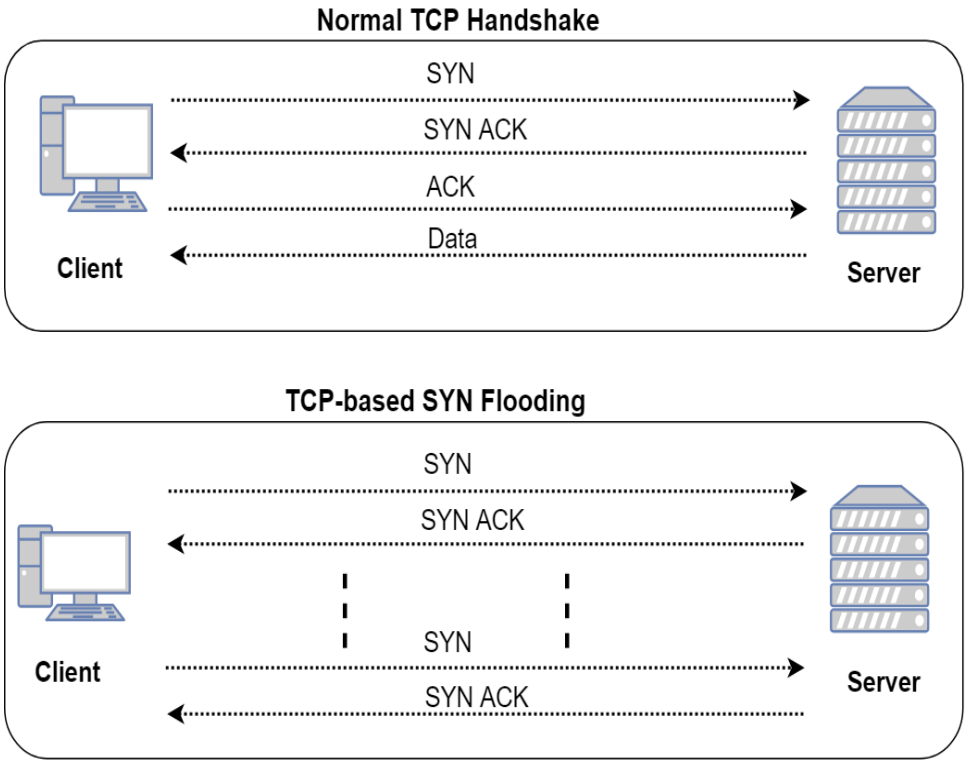

Figure 5.1: Typical TCP handshake (top) and SYN flooding attack (bottom)

message from the client (within a given timeout interval). It is only once the client replies back with an ACK message that the TCP session is deemed as successfully established and data can be exchanged.

Now as per the above, TCP session establishment is very susceptible to SYN flooding attacks, i.e., where an adversary sends a large number of half-open connections to the server without sending closing ACK packets, also shown in Figure 5.1. These floods can lead to resource exhaust at the server, preventing legitimate new connections from being established. Moreover, in some cases many distributed adversarial hosts can even send a large amount of

SYN packets using spoofed IP addresses (i.e., DDoS attacks), making it even more difficult for a server to identify the sources of an attack.

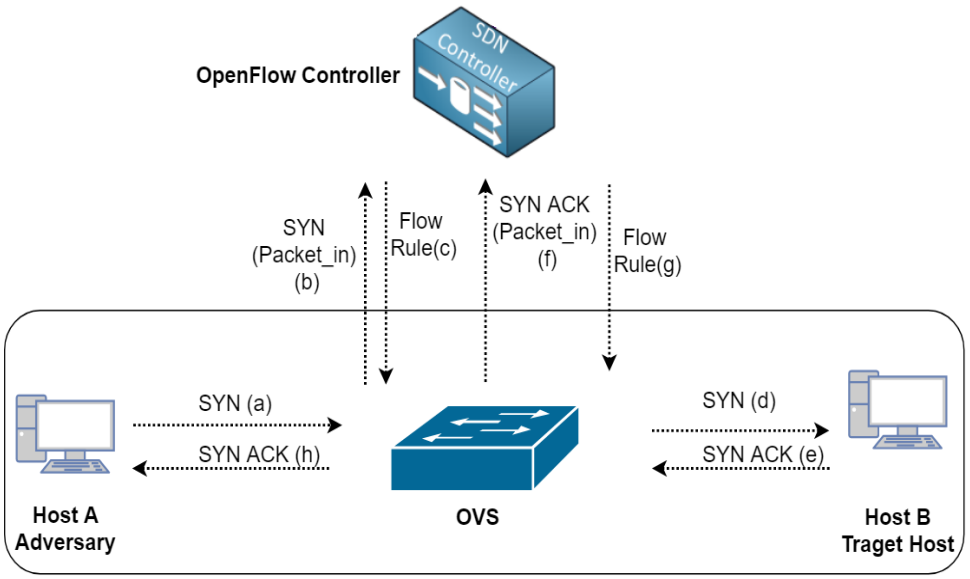

Figure 5.2: SYN flooding attack in SDN

5.1.2 Threat and Attack Model in SDN Environment

The threat model here assumes both data and control plane vulnerably to TCP SYN flooding attacks. Furthermore, it is also assumed that (1) an attacker is capable of sending a large amount of TCP SYN-based traffic to an OpenFlow network, and (2) adversarial traffic is comprised of a mixture of both legitimate and malicious packets, making it difficult to differentiate. Accordingly, Figure 5.2 shows a realistic scenario that illustrates how an attacker can flood the SDN infrastructure with TCP SYN packets. Here, the Open vSwitch device receives an incoming packet stream containing a mixture of legitimate (normal user traffic) packets and malicious SYN flooding packets (from the adversary host). It is assumed

that the single adversary host is capable of producing a large number of SYN packets to saturate the switching device.

Now consider the detailed interactions between the end hosts, switches, and SDN controller shown in Figure 5.2. Foremost, the adversary host (Host A) sends SYN flooding packets (step a) and the receiving switch checks its flow table for a matching forwarding rule. If no flow rule matching the destination IP address is found, i.e., table-miss, the switch forwards it to the SDN controller (step b) in the form of $Packet\_In$ message. The SDN controller processes this $Packet\_In$ request and searches for the corresponding path to the target host (assuming that there is at least one path available). It then inserts the flow rule into the switch's flow table (step c). Next, the switch forwards the SYN packet to the target host, Host B (step d). Finally, when Host B receives the SYN packet, it allocates a new buffer for this request and replies with a SYN_ACK packet (step e). Similarly, once the destination switch receives the SYN_ACK message, it again forwards it to the SDN controller for flow rule query (step f). Upon receiving this $Packet\_In$ message from the switch, the SDN controller once again computes the return route and inserts it in the switch flow table (step g). Finally, the switch sends a SYN_ACK message to Host B (step h). Given this handshake-based communication model, an adversary can readily insert false source IP addresses and make the SDN controller insert spurious new forwarding rules (step g). Namely, the forwarded SYN_ACK packets to Host B will be based on invalid IP addresses.

In light of the above, if the adversary host transmits a large number of "spoofed" SYN packets, it can easily flood the resources at the target host. More importantly, the adversary can also overwhelm switch lookup tables with spurious flow rules, consuming precious memory. Furthermore, SYN floods can impose high computational workload at the SDN controller, i.e., $Packet\_In$ processing, and also saturate bandwidth usage at the control plane due to excessive messaging, i.e., $Packet\_In$ messages and flow rules.

5.1.3 Motivation and Problem Scope

SDN environments can face many specialized threats from malicious adversaries. Hence the simple pre-placement of flow rules in data plane switches to dismiss any non-matching flows is not practical. Instead, SDN control implementations typically consist of different processing policies since network traffic is usually very dynamic. For example, policies can differ according to varying network conditions or user types [64]. Indeed, dynamic processing policies can render controller applications susceptible to security SYN flooding threats because such policies require continuous updates during the data layer transition [64]. Now recent versions of the OpenFlow protocol also allow the SDN controller to receive information about data layer transition via incoming $Packet\_In$ messages. Hence when a SYN flooding attack occurs, the controller's resources can easily be exhausted.

In light of the above, the first objective here is to try to maintain the core functionality of the SDN infrastructure during SYN flooding saturation attacks. In particular, this is done via direct installation of flow rules in SDN switching devices in real-time. Furthermore,

another objective is to try to process table-miss flows without harming normal/legitimate user traffic flows. In particular, simply dropping all table-miss traffic is not acceptable as it may adversely impact legitimate users as well. Instead, whenever a SYN flooding attack takes a place, the table-miss traffic will be sent to the corresponding data layer cache prior to communication with the SDN control layer (via $Packet\_In$ messages).

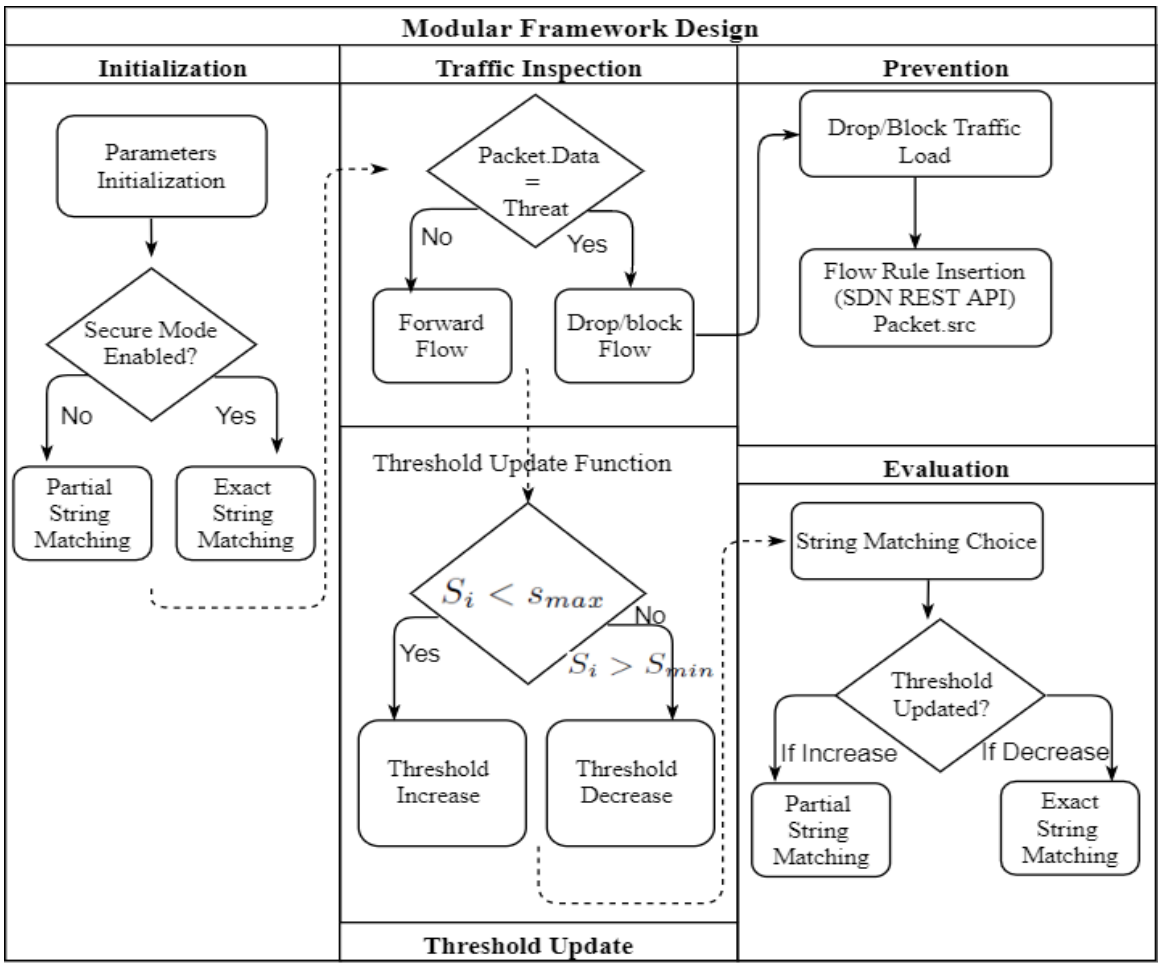

Figure 5.3: Architecture of proposed SYNGuard framework

5.2 Modeling and Framework Design

In order to address SYN flooding attacks in SDN settings, a novel protocol independent IDPS solution is now presented, i.e., termed as SYNGuard. The proposed scheme runs as an application at the SDN controller and is comprised of several modules, as shown in Figure 5.3. As noted earlier, SYNGuard makes exclusive use of SDN capabilities to monitor and collect statistics from incoming data plane traffic and exact flow routing rules/responses. The overall placement of this solution (in relation to the Floodlight controller and Open vSwitch platform) is also shown in Figure 5.4. Namely, this scheme uses Open vSwitch Switched Port Analyzer (SPAN) ports to mirror all incoming flows on the corresponding device (for analysis). Precisely, SPAN ports support communication traffic between the application and data layers of the SDN setup. Overall, this positioning is critical for effective identification and threat mitigation of SYN flood traffic, as shown in Figure 5.4. Now the architectural framework in Figure 5.3 consists of four key modules; (1) *Initialization*, which sets up initial operation of the switches, (2) *Inspection*, which examines the traffic received through the raw operating system socket (e.g., SPAN port), (3) *Mitigation*, which tries to limit false positive detection events, and (4) *Threshold_Evaluation*, which dynamically adjusts the attack detection threshold according to the current record of attacks (based upon the SDN controller view).

Now in order to determine whether incoming traffic constitutes a TCP SYN flood attack, the pattern of incoming flows over a time frame is identified. Namely, when the

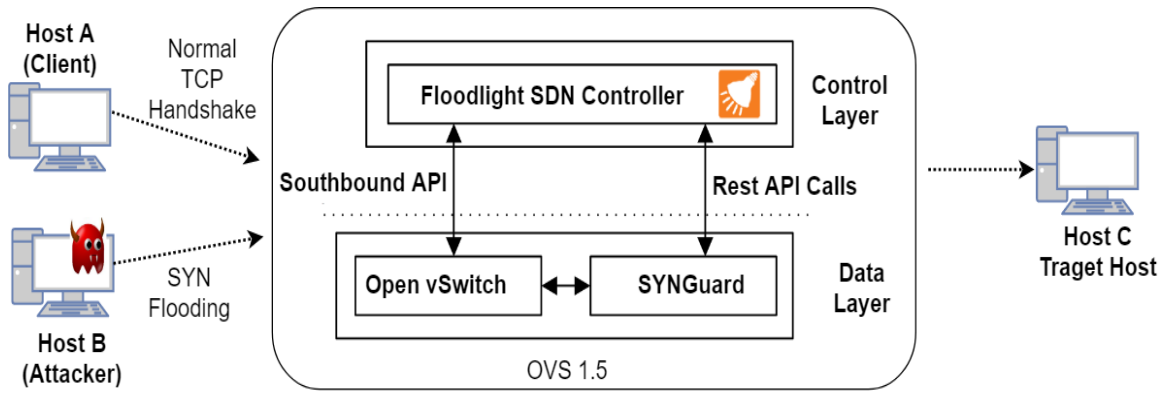

Figure 5.4: The operational placement of proposed IDPS for traffic flow

current threshold value is surpassed by the quantity of arriving flows, the inspection module in the framework detects/flags a SYN flood attack and generates an alert message. Contrary to existing IDPS solutions, the dynamic threshold mechanism in the proposed framework also relies on the state of the inspection module and frequency of generated alerts.

Overall, SYNGuard is (1) a lightweight design that uses a modular-based filtering technique, (2) runs as an independent network application that utilizes SPAN interfaces on Open vSwitch platforms, (3) adds minimal overhead and traffic processing latency, and (4) operates transparently to end hosts and SDN-enabled applications. The latter ensures improved applicability to larger networking environments. Further details are now presented.

5.2.1 Adaptive Detection Threshold and Signature Structure

In the context of DoS/DDoS attack detection, threshold values generally refer to the rate of attack events per second. Specifically, a detection threshold represents the rate at which an IDPS raises an alert for an attack. For example, if the event threshold is set to 70 packets per second, an alert will be generated once the incoming packets from a source exceeds this bound. Hence selecting an appropriate detection threshold value is critical in order to provide efficacious response to DoS threats while reducing false alerts (i.e., threshold value too high or too low). To address these challenges, a dynamic self-adjusting detection threshold is proposed for SYNGuard. Consider some further details here.

Foremost, it is assumed that legitimate user flows generate TCP SYN requests at steady intervals. Namely, the number of SYN connection requests range from $[r_1\ ,\ r_2\ ]$, whereas the number of flows per SYN connection is $n \geq 3$ (i.e., three-way handshake exchange). Additionally, all incoming SYN flows are assumed to follow a Poisson arrival distribution with an average of λ flows/second. Hence in order to compute the number of packets per SYN connection, n, the SDN controller sends a Flow Statistics Collection (FSC) request via a OpenFlow *Read_State* message to switches with the matched field set equal to the IP addresses in the flow table. The value of n can then be estimated by using the OpenFlow *Packet_Count* parameter, which is a pre-defined metric that tracks the number of incoming flows based upon the switch flow table, i.e.,

$$n = \min_{\forall f_i \in F} (pc_i) \tag{5.1}$$

where F and pc_i are the set of flow tables and minimum $Packet\_Count$ value, respectively.

Now consider the dynamic threshold update mechanism for SYN flood detection. Here, the ubiquitous TCP Additive Increase Multiplicative Decrease (AIMD) window size control mechanism is adopted. Namely, let Ψ_I and Ψ_D represent the updating functions for threshold increase and decrease, respectively. Now in the default TCP AIMD mechanism,

$$\Psi_I = S + 1, \tag{5.2}$$

and

$$\Psi_D = \frac{S}{2}, \tag{5.3}$$

where S is the threshold value bounded by $S_{min} \leq S \leq S_{max}$, i.e., S_{min} and S_{max} represent the minimum and maximum threshold ranges, respectively. Leveraging from the above, the SYNGuard solution introduces a modified, improved AIMD mechanism that dynamically adjusts the increase and decrease levels based upon the event state of the SYN flood threat. Namely, now the increase and decrease operations (for the i-th interval) are given by:

$$\Psi_I = S_i + \alpha, \tag{5.4}$$

and

$$\Psi_D = \frac{S_i}{\beta}, \tag{5.5}$$

where $S_{min} \leq S_i \leq S_{max}$, and α and β represent the threshold increase and decrease values, respectively. Furthermore, S_0 is manually initialized at a default threshold value at startup.

As a further improvement, a single update function is also defined for the detection threshold. This function is continuously invoked by SYNGuard to update the current threshold according to both legitimate and total traffic rates. Namely, the two update functions, Ψ_I (Equation 5.4) and Ψ_D (Equation 5.5), are now replaced by a next-round threshold value, $S(i+1)$, such that:

$$S(i+1) = P_1 + P_2 \tag{5.6}$$

where P_1 and P_2 are defined as:

$$P_1 = p\left(\frac{n}{T} \geqslant \mathrm{S(i)}\right).\Big[q(i).\Psi_I\Big(S(i)\Big) + \Big(1 - q(i)\Big).\Psi_D\Big(S(i)\Big)\Big] \tag{5.7}$$

and

$$P_2 = p\left(\frac{n}{T} \leqslant \mathrm{S(i)}\right).\Big[q(i).\Psi_I\Big(S(i)\Big) + \Big(1 - S(i)\Big).\Psi_D\Big(S(i)\Big)\Big] \tag{5.8}$$

Substituting P_1 and P_2 into Equation 5.6, and simplifying gives a threshold update function as follows:

$$S(i+1) = q(i).\Psi_I\Big(S(i)\Big) + \Big(1 - q(i)\Big).\Psi_D\Big(S(i)\Big) \tag{5.9}$$

where $q(i).\Psi_I(S(i)) + (1-q(i)).\Psi_D(S(i))$ is assumed to be an integer (or rounded otherwise), and $p(\frac{n}{T} \geqslant \mathrm{S(i)})$ and $p(\frac{n}{T} \leqslant \mathrm{S(i)})$ are the probabilities that both events $\frac{n}{T} \geqslant \mathrm{S(i)}$ and $\frac{n}{T} \leqslant \mathrm{S(i)}$ occur respectively.

Now carefully note that packet signature matching may not always be the best strategy to identify SYN flood attacks, i.e., since malicious hosts may be capable of injecting any arbitrary data bytes into packet payloads. As a result, signature matching can actually be defeated by an adept adversary. Moreover, certain types of TCP SYN flooding attacks may not contain proper packet payload data, e.g., low-profile SYN flood attacks. Hence, the proposed framework also tries to match packet header information (instead of packet payload data) in order to improve anomaly detection performance. Specifically, this approach can help increase the effectiveness of threat signature matching and detection and can also handle low-profile SYN attacks (with no particular data format in the packet payloads).

The overall pseudo code of traffic inspection is shown in Figure 5.5, where the header information of packet is checked using string matching techniques. Here the variables T, Th, and S_i represent the timestamp, threshold value, and list of strings (i.e., for string matching purposes), respectively. In addition, $S(.)$ is defined as a counter function to track the number of packets per source address (for threshold comparison). Now if an ICMP packet is received, the packet counter, $S(.)$, is updated and its header information is compared with the existing signatures, i.e., string search. Hence an alert is generated based upon the current threshold and string matching result, i.e., anomalous packet load. Lastly, the threshold update function is invoked for appropriate adjustment of current attack detection rate.

```
1: Set P = PACKET_IN
2: Set Th = threshold_value
3: while True
4:    T = Time_frame[Timestamps]; S_0 = P/T
5:    if P.Type == ICMP
6:       Calculate n/T
7:       QP.SRC_ADDR++
8:       if P.SRC_ADDR NOT IN S
9:          S(P.SRC_ADDR) = T
10:   else
11:      if T − S(P.SRC_ADDR) > Th AND Si < Smax
12:         if Q(P.SRC_ADDR) > Th
13:            Increase Th
14:         else
15:            S(P.SRC_ADDR) = T
16:      else
17:         Decrease Th if Si > Smin
18: return 0
```

Figure 5.5: Proposed algorithm for the detection threshold update

5.2.2 Overall Framework Functionality

Now once SYNGuard has been initialized, the Inspection module (Figure 5.6) monitors and classifies all TCP SYN requests, i.e., established TCP handshakes, while recording source-side connection timeouts, i.e., failed connections. Once a connection request (i.e., first SYN packet) is received at the data layer, the Open vSwitch searches for a matching TCP ACK forwarding rule in its flow table or inquires about it from the SDN controller. Carefully note that the Open vSwitch device also verifies whether the received packet type is a TCP SYN (prior to validating the completion of the TCP handshaking session) in order to identify unsolicited packets.

The overall finite-sate machine for the SYNGuard system is also shown in Figure 5.6.

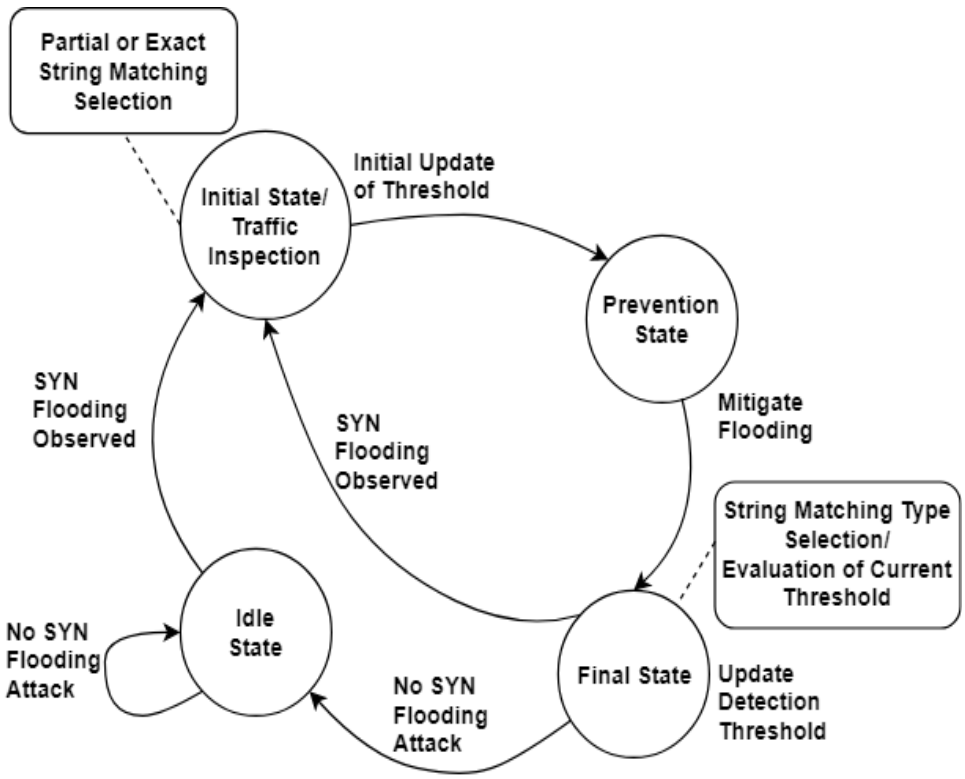

Figure 5.6: SYNGuard states

Here the initial state specifies all the input and global variables used in the inspection and mitigation modules and jump starts the packet inspection process. Once a malicious TCP SYN event is detected in the Inspection state, a string matching operation is carried out, followed by alert generation. Next, if the current threshold value of detected events is exceeded, the system moves to the Prevention state to mitigate the attack. Lastly, the system also invokes the threshold update function to update the current detection threshold.

Finally, as shown in the architecture in Figure 5.3, once a flow is received at the respective Open vSwitch port, the switch will transition from the Idle state to the Inspec-

tion state, i.e., to start traffic inspection prior to forwarding it to the potential target. It is expected that the network administrator will specify/enter a default threshold value at startup as well. Finally, a secure mode option is also supported in order to allow the IDPS solution to conduct more comprehensive deep packet (payload) inspection using exact string matching, i.e., in the case of sensitive data. However, if the switch is located in a trusted region, the IDPS will only perform partial string matching.

Overall, switches will continuously inspect and examine all incoming flows and keep track of malignant adversary traffic. Based upon this, threat detection thresholds will be updated in a dynamic manner (as per the current trace and intensity of illegitimate traffic). For example, if a sequence of flows is found to be legitimate, the switch will decrease the threshold S_i by a value β and also select exact string matching filters (or partial ones). Alternatively, if a particular flow is found to be malicious, the controller will forward it to the prevention module (Figure 5.3) to either drop or block it.

5.3 Performance Evaluation

The performance of the proposed SYNGuard solution is now tested in a real-world heterogeneous testbed setup with various networking capabilities. Again the NSF GENI facility is used here, and the proposed topology slice is shown in Figure 5.7. Namely, this setup consists of two end hosts, Host A and Host B, SDN controllers, and four Open vSwitches. In particular, Host A represents the adversary which transmits a mixture of legitimate and malicious traffic to the target at Host B. Meanwhile, hosts O1-O4 represent Open vSwitches

running the IDPS solution (in addition to performing their essential traffic forwarding operations). Finally, hosts C1 and C2 represent the SDN controllers running the Floodlight controller software. Namely, controller C1 manages Open vSwitches O1 and O2, and controller C2 manages Open vSwitches O3 and O4, respectively. Lastly, the dashed and solid lines represent the control and data plane links, respectively. Although a relatively simple topology is used to evaluate the performance of SYNGuard scheme here, the solution can readily be used in larger and more complex networking environments.

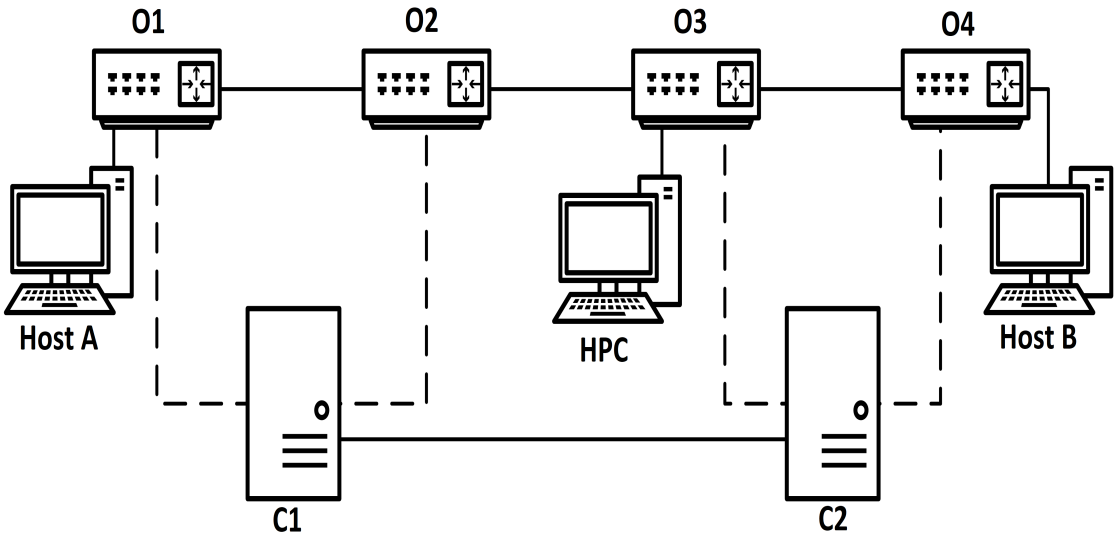

Figure 5.7: GENI testbed topology

5.3.1 Traffic Generation and Event Rules Implementation

Initial tests are done to evaluate system response time to threats under simulated real-world networking conditions. Namely, two network traffic intent types are considered here, i.e., adverse (malicious) and normal (benign). Packets from these flows are then gener-

ated and mixed into the network. Furthermore, both the iPerf and the Distributed Internet Traffic Generator (D-ITG) toolkits are used in tandem to generate packet traffic and quantify/control it across the network. Note that iPerf also allows saturation of edge network links, which can be used as a baseline technique to evaluate response efficiency to saturation attacks. Additionally, both the Zeek and Snort IDS toolkits are also deployed here for rule-based detection. Namely, pre-defined detection rules are manually added or configured, i.e., if they are already implemented in particular version. However, in order to ensure proper performance comparison, the only rules used for Zeek and Snort are those for SYN flood attack detection and mitigation. Since both of these tools can handle a broader range of network threats, comparing their associated overheads with other rules enabled will not be meaningful (and hence they are offloaded or disabled). Accordingly, the sample scripts in Figures 5.8 and 5.9 also show the actual detection rules for SYN-based flood traffic in the Snort and Zeek IDS tools, respectively. Namely, in Figure 5.8 Snort raises a TCP-SYN flagged threat alert based upon a default threshold value, e.g., whenever 70 or more packets arrive within a 10 second interval. Meanwhile in Figure 5.9, Zeek uses a specified threshold, n, which can be manually adjusted during configuration setup. This parameter counts the number of failed TCP connections and raises an alert flag when this count is exceeded.

5.3.2 Performance Results

Overall, effective IDPS solutions play a critical role in countering malicious attack traffic. As noted, adverse packet flows can result in end host resource exhaustion, link and

```
alert tcp any any -> $HOME_NET 80(flags:S;
msg:"Possible TCP DoS is Detected!";
flow: stateless;
detection_filter: track by_dist, count 70,
seconds 10; sid 10001;rev:1;)
```

Figure 5.8: SYN flooding rule sample in Snort

```
Global attempts: table[$addr$] of count $&$ default=0;
event connection_rejected(c: connection){
    local source = c$id$orig_h;   #Get source address;
    local n = ++attempts[source]; #Increase counter;
    if (n=some_threshold);        #Check for threshold;
    Notice(...);                  #Alarm;}
```

Figure 5.9: SYN flooding rule sample in Zeek, n is the threshold

switch saturation, as well as other bottlenecks. Therefore proper flow inspection is critical for limiting resource exhaust. Along these lines, the performance of the SYNGuard solution is examined with regards to the inspection times, threat response times (mitigation), threat identification accuracy, and performance overheads. Furthermore, comparisons are also made with some existing well-established solutions, including Snort and Zeek.

Foremost, inspection time for incoming flows is a critical parameter that directly impacts malicious flow response times. Namely, when flows are captured by the IDPS, the related packet information must be inserted into a buffer while awaiting inspection. Expectedly, this waiting time will add to the overall time required for mitigating the attack if an alert is raised. Hence in order to evaluate the inspection times of the proposed solution,

Table 5.1: Average inspection and mitigation times and system load

Traffic Load (1,000s of packets)		100	200	500
Metric	IDPS			
Inspection (Sec.)	Proposed IDPS	0.0031	0.0068	0.0117
	Snort	0.0041	0.0059	0.0145
	Zeek	0.0073	0.0102	0.0130
Mitigation (Sec.)	Proposed IDPS	0.0029	0.0081	0.0112
	Snort	0.0045	0.0073	0.0130
	Zeek	0.0076	0.0100	0.0128
System (%)	Proposed IDPS	8.04	12.70	15.09
	Snort	9.02	15.08	19.17
	Zeek	17.53	26.05	34.19

both the Snort and Zeek solutions are implemented with near equivalent configurations as used in the SYNGuard scheme. Specifically, for evaluation fairness, each IDPS solution is configured with similar threat detection signatures.

To gauge inspection times, a communication scenario is established between two hosts using the *hping*3 tool. Specifically, smaller-sized packets are generated at a high rate in a very small time window (e.g., one nanosecond) in order to achieve a high network link saturation. Note that the proposed SYNGuard solution automatically adjusts its detection threshold, whereas this value is static in both the Zeek and Snort schemes (and hence needs to be manually set at initialization time). Hence these static detection thresholds are set to default values of 10 seconds each. Experimental runs are also conducted for 15 successive trials in an automated manner, and the results are averaged, as shown in Figure 5.10 and Table 5.1. Overall, these findings demonstrate that the proposed solution notably outperforms both Snort and Zeek. For example, Zeek gives almost 100% higher inspection times, whereas Snort gives about 10-20% higher latencies. However, for some randomized runs, Snort can

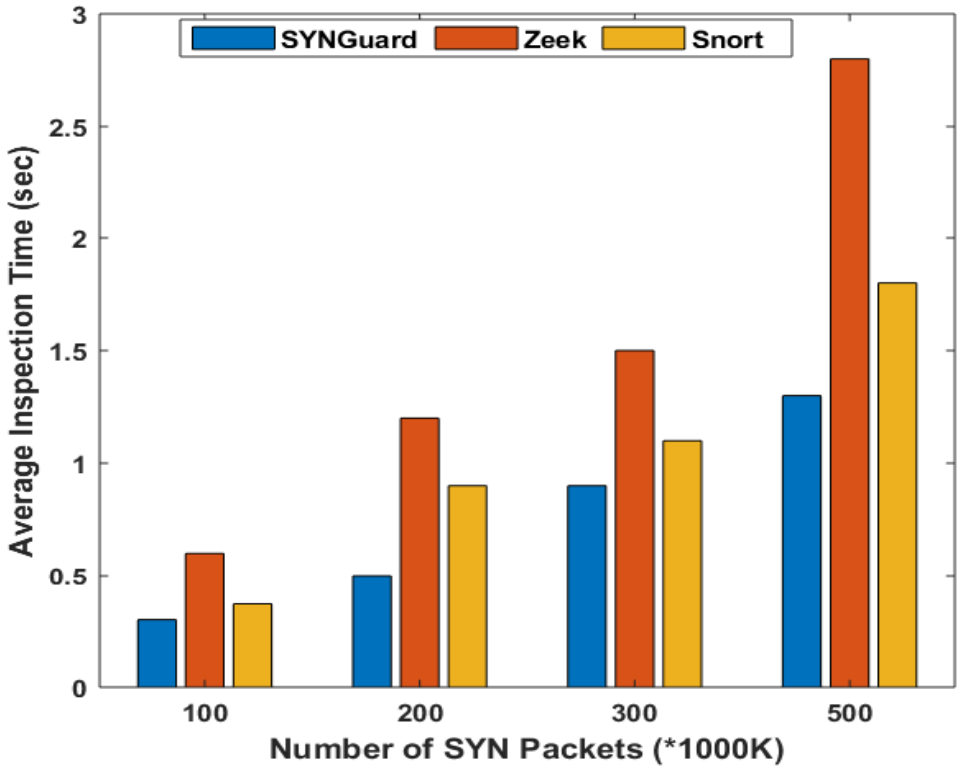

Figure 5.10: Average inspection times for SYN flooding (10 sec threshold, Snort and Zeek)

match (but not exceed) the performance of the SYNGuard scheme.

Next, consider mitigation time, which is defined as the response time to handle threatening adversary flows. Namely, once a specific flow has been detected and flagged as malicious, the mitigation time is defined as the time between raising an attack alert and taking necessary corresponding action, e.g., drop, block, etc. Indeed, this delay is a critical factor in assuring the availability of key operational resources in SDN environments. Now even though the detection attributes (for malicious traffic) in each IDPS scheme are unique, the mitigation time for the SYNGuard scheme is still compared against Snort and Zeek. Specifically,

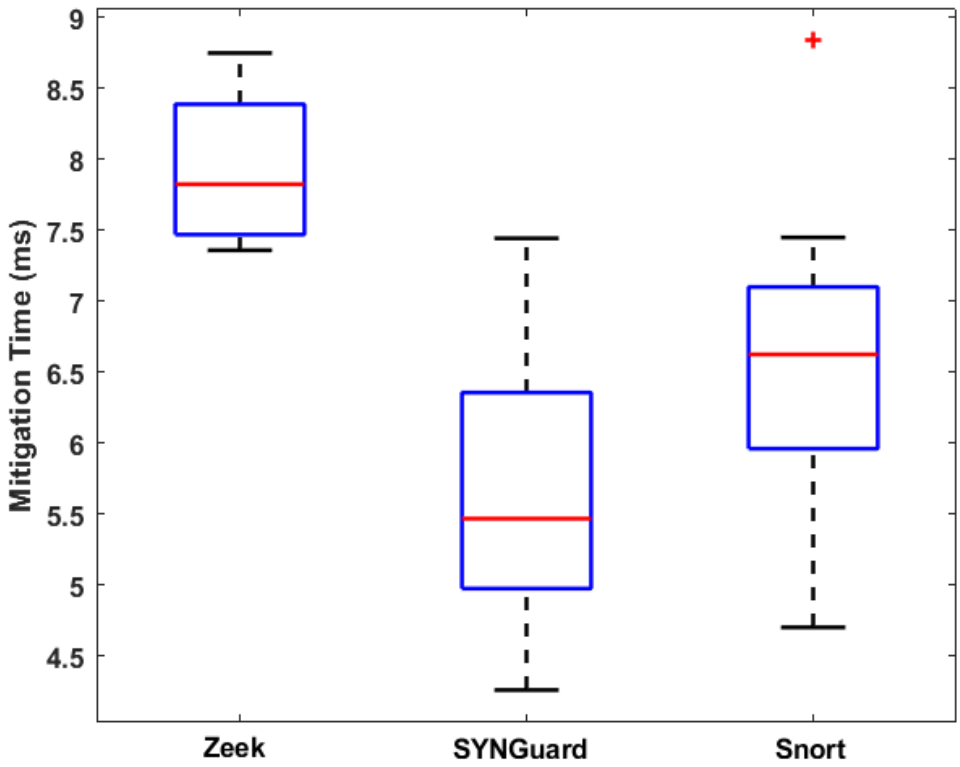

Figure 5.11: SYN flood mitigation time (100,000 SYN flagged packets)

this value is measured as the time between attack initiation and attack rectification.

Along these lines, Table 5.1 summarizes the average mitigation times for the three IDPS solutions using the same experimental configuration detailed earlier, i.e., threshold initialization, threat detection signature, etc. Note that these results are averaged over 15 runs. Overall, the findings confirm that the SYNGuard solution provides relatively-similar mitigation times to Snort, but significantly smaller than Zeek. It is assumed that each IDPS scheme initiates blocking actions against harmful IP addresses once an attack alert is generated. Furthermore, Figure 5.11 also plots the attack mitigation times for all schemes

(with Snort and Zeek using their default 10 second intervals). Again, these findings show that the SYNGuard scheme outperforms both Snort and Zeek, i.e., average mitigation times are about 40% lower than Zeek and 20% lower than Snort.

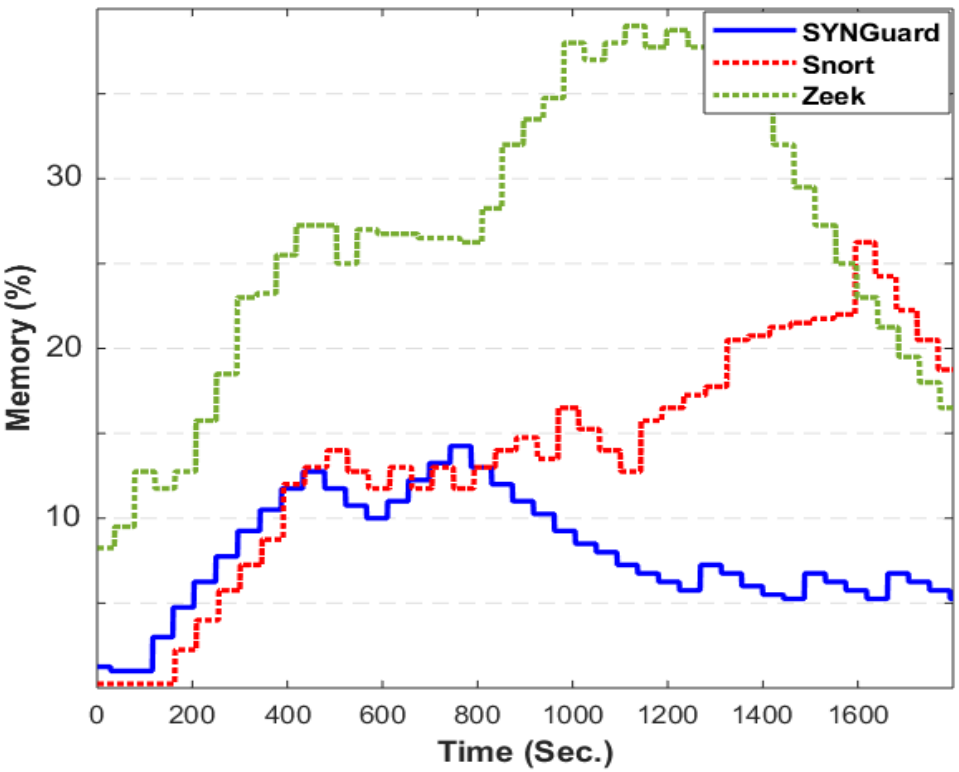

Figure 5.12: Memory utilization (100,000 SYN flagged packets)

Now excessive inspection operations can also lead to resource drainage on the platforms running the IDPS software (due to overheads associated with examining packets). Hence to gauge system overheads, Figure 5.12 also plots the memory utilization for each IDS scheme for the case of a single attack with 100,000 SYN flagged packets. Here both the Snort and Zeek solutions are initialized to use their default detection threshold (10 seconds). Carefully note that link saturation also occurs in this attack scenario. These results

clearly show lower memory utilization with the proposed SYNGuard scheme, particularly in comparison to the Zeek solution. Note that Table 5.1 also summarizes the measured system load in terms of average resource utilization for a variety of network flooding attacks with 100,000, 200,000, and 500,000 SYN flagged attack packets (from adversary Host A to target Host B, Figure 5.7).

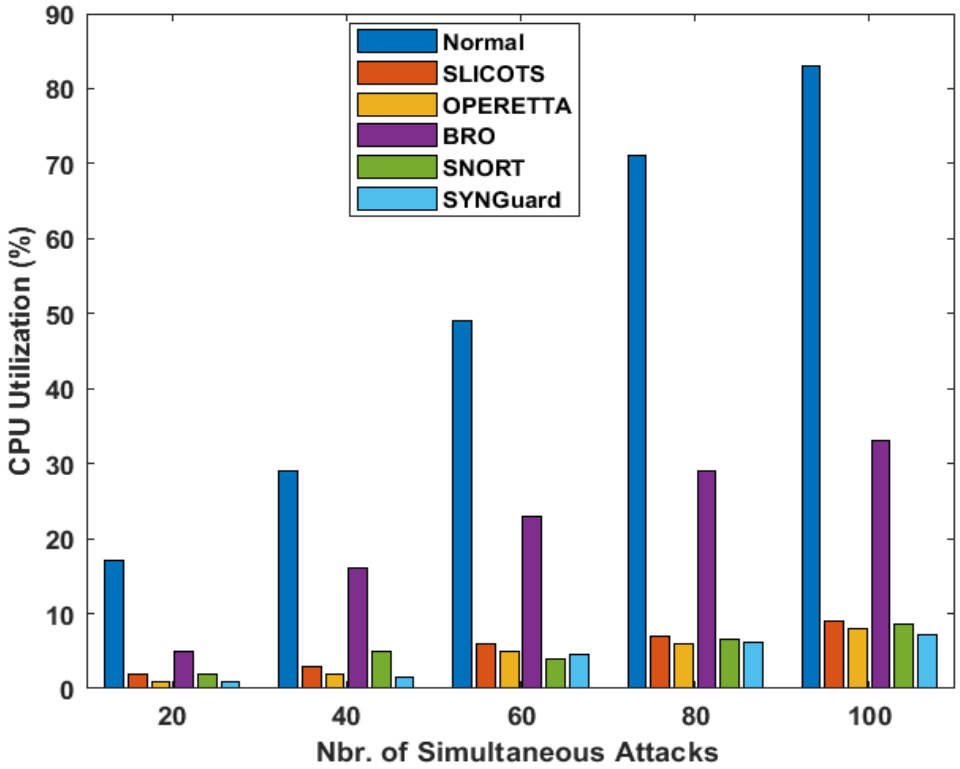

Figure 5.13: CPU utilization for varying simultaneous attack sessions (same source)

Finally, Figure 5.13 plots the CPU utilization overhead for SYNguard, Snort, Zeek, normal SDN (unprotected with no IDS), SLICOTS [23], and OPERETTA [24]. In particular, SLICOTS and OPERETTA are state-of-art IDS solutions (developed by R. Mohammadi et al. [23] and S. Fichera et al. [24], respectively) to prevent SYN flooding attacks in

SDN networks. These schemes work by surveiling failed TCP SYN requests and blocking malicious adversary hosts. Specifically, SLICOTS is implemented as an extension of the OpenDayLight controller software, whereas OPERETTA is developed as an SDN framework using the POX controller. Overall, the results in Figure 5.13 show that the SYNGuard solution gives the lowest CPU utilization of all, i.e., many factors lower than the Zeek toolkit. However the Snort solution performs relatively well, and closely tracks SYNGuard performance. It is important to note that both SLICOTS [23] and OPERETTA [24] can also prevent control plane saturation. Regardless, SYNGuard still yields slightly less impact on CPU overheads than OPERETTA, Snort, and Zeek. Like SLICOTS, SYNGuard also benefits from temporary injection of flow rules in Open vSwitch devices. Hence the high rate of malicious SYN flooding traffic has less of an impact on the control-data channel, which further decreases overall resource overheads.

Chapter 6 : Conclusions and Future Work

This research dissertation focuses on QoS and security support in emerging ICT-based networks using SDN technologies. First, Chapter 2 presents a background survey of some existing work in related areas. Next, Chapter 3 details a novel scheme for end-to-end path latency management in SDN settings. Building upon this, Chapter 4 presents further priority-based mechanisms for effective traffic management. Finally, Chapter 5 addresses the critical topic of SDN security and tables a dynamic threshold-based countermeasures solution for detecting and mitigating data-control channel saturation attacks. Overall conclusions from this dissertation effort are now presented along with some discussions on potential future research directions.

6.1 Conclusions

Overall, some of the key contributions and findings from this study are as follows:

- To the best of the author's knowledge, this dissertation presents one of the first studies on using SDN to develop systematic schemes to reduce end-to-end path delay and latency associated with controller-switch communication. Namely, the proposed solution uses a time series (TS) estimation approach along with adaptive $A*$ path computation to select end-to-end paths with reduced delays.

- This work is one of the first to introduce a QoS-aware scheme that leverages globalized SDN control in conjunction with priority-based queueing and TE techniques. Namely, the solution uses multiple queues in each switch port to support different data plane traffic priority levels while preempting control plane traffic.
- To improve the security posture of SDN operation, this work also presents a novel IDPS solution to effectively detect and mitigate data-control plane saturation attacks, i.e., TCP SYN floods. Namely, this solution implements a dynamic threshold event detection strategy to lower inspections, i.e., based upon the Additive Increase Multiplicative Decrease (AIMD) mechanism.

Furthermore, the NSF Global Environment for Networking Innovations (GENI) facility, a heterogeneous and at-scale real-world testbed, is also used to prototype and evaluate the performance of all mechanisms in this dissertation study. Whenever applicable, appropriate comparisons and analyses are also done with existing, well-established security schemes. Overall, detailed performance evaluation studies have demonstrated that these solutions outperform existing counterpart schemes in term of accuracy and efficiency (with regard QoS and security improvements in SDN-enabled communication systems).

6.2 Future Work

Overall, this dissertation effort presents some novel solutions for improving QoS support and security in evolving ICT networks via the integration of SDN-based technology. Specifically, a key objective has been to reduce end-to-end delays for stringent traffic flows,

while taking into account switching latency and controller response overheads. Hence future efforts can consider the extension of these mechanisms into broader multi-domain SDN setups operating with multiple SDN controllers. Furthermore, other QoS strategies can also be implemented here, e.g., such as priority-based routing and path selection for emergency response services.

Now from the SDN security perspective, this study has presented inspection and detection techniques to resolve data-control plane saturation attacks using dynamic thresholding. Building upon this, future efforts can also look at identifying other potential security threats against both the data and control planes which require Deep Packet Inspection (DPI). For example, these can include Link Discovery Service (LDS) exploitation, TCP-based packet spoofing, and TCP reply-enabled DoS threats.

References

[1] Ying He, F Richard Yu, Nan Zhao, Victor CM Leung, and Hongxi Yin. Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach. *IEEE Communications Magazine*, 55(12):31–37, 2017.

[2] Shaibal Chakrabarty and Daniel W Engels. A secure iot architecture for smart cities. In *2016 13th IEEE annual consumer communications & networking conference (CCNC)*, pages 812–813. IEEE, 2016.

[3] Rahim Masoudi and Ali Ghaffari. Software defined networks: A survey. *Journal of Network and computer Applications, Elsevier*, 67:1–25, 2016.

[4] Hamid Farhady, HyunYong Lee, and Akihiro Nakao. Software-defined networking: A survey. *Computer Networks, Elsevier*, 81:79–95, 2015.

[5] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, and William Snow. Onos: towards an open, distributed SDN OS. In *Proceedings of the third ACM workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 1–6, 2014.

[6] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review (CCR)*, 38(2):69–74, 2008.

[7] Opencontrail. `http://www.opencontrail.org`. [Accessed March 2020].

[8] Peter Saint-Andre, *et al.* Extensible messaging and presence protocol (xmpp): Core. 2004.

[9] Andrew Mckeown, Habib Rashvand, Tony Wilcox, and Paul Thomas. Priority SDN controlled integrated wireless and powerline wired for smart-home internet of things. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1825–1830. IEEE, 2015.

[10] Walaa F Elsadek and Mikhail N Mikhail. Inter-domain mobility management using SDN for residential/enterprise real time services. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 43–50. IEEE, 2016.

[11] Issa Khalil, Abdallah Khreishah, and Muhammad Azeem. Consolidated identity management system for secure mobile cloud computing. *Computer Networks*, 65:99–110, 2014.

[12] Tommy Chin, Mohamed Rahouti, and Kaiqi Xiong. Applying software-defined networking to minimize the end-to-end delay of network services. *ACM SIGAPP Applied Computing Review*, 18(1):30–40, 2018.

[13] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[14] Hilmi E Egilmez, S Tahsin Dane, K Tolga Bagci, and A Murat Tekalp. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Proceedings of the Asia Pacific signal and information processing association annual summit and conference (APSIPA ASC)*, pages 1–8. IEEE, 2012.

[15] Sachin Sharma, Dimitri Staessens, Didier Colle, David Palma, Joao Goncalves, Ricardo Figueiredo, Donal Morris, Mario Pickavet, and Piet Demeester. Implementing quality of service for the software defined networking enabled future internet. In *The European Workshop on Software Defined Networking (EWSDN)*, pages 49–54. IEEE, 2014.

[16] Abdul Basit, Saad Qaisar, Syed Hamid Rasool, and Mudassar Ali. SDN orchestration for next generation inter-networking: A multipath forwarding approach. *IEEE Access*, 5:13077–13089, 2017.

[17] Raphael Durner, *et al.* Performance study of dynamic QoS management for openflow-enabled SDN switches. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 177–182. IEEE, 2015.

[18] Seyeon Jeong, *et al.* Application-aware traffic engineering in software-defined network. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 315–318. IEEE, 2017.

[19] Keqiang He, Junaid Khalid, Sourav Das, Aditya Akella, Erran Li Li, and Marina Thottan. Mazu: Taming latency in software defined networks. Technical report, 2014.

[20] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, volume 15, pages 8–11, 2015.

[21] Rui Wang, Zhiping Jia, and Lei Ju. An entropy-based distributed DDoS detection mechanism in software-defined networking. In *Proceedings of IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 310–317, 2015.

[22] Tri-Hai Nguyen and Myungsik Yoo. Analysis of link discovery service attacks in SDN controller. In *2017 International Conference on Information Networking (ICOIN)*, pages 259–261. IEEE, 2017.

[23] Reza Mohammadi, Reza Javidan, and Mauro Conti. Slicots: An SDN-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2):487–497, 2017.

[24] Silvia Fichera, Laura Galluccio, Salvatore C Grancagnolo, Giacomo Morabito, and Sergio Palazzo. Operetta: An openflow-based remedy to mitigate tcp synflood attacks against web servers. *Computer Networks*, 92:89–100, 2015.

[25] Mohamed Rahouti, Kaiqi Xiong, Tommy Chin, Peizhao Hu, and Diogo De Oliveira. A preemption-based timely software defined networking framework for emergency response traffic delivery. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 452–459. IEEE, 2019.

[26] Mohamed Rahouti, Kaiqi Xiong, Tommy Chin, and Peizhao Hu. Sdn-ers: A timely software defined networking framework for emergency response systems. In *International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC)*, pages 18–23. IEEE, 2018.

[27] Haoqi Ni, Mohamed Rahouti, Aranya Chakrabortty, Kaiqi Xiong, and Yufeng Xin. A distributed cloud-based wide-area controller with SDN-enabled delay optimization. In *Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2018.

[28] Tommy Chin, Kaiqi Xiong, and Mohamed Rahouti. SDN-based kernel modular countermeasure for intrusion detection. In *International Conference on Security and Privacy in Communication Systems*, pages 270–290. Springer, 2017.

[29] End to-end delay minimization approaches using software-defined networking. End-to-end delay minimization approaches using software-defined networking. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems (RACS)*, pages 184–189. ACM, 2017.

[30] Danny Yuxing Huang, Kenneth Yocum, and Alex C Snoeren. High-fidelity switch models for software-defined network emulation. In *SIGCOMM workshop on Hot topics in software defined networking*, pages 43–48. ACM, 2013.

[31] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W Moore. Oflops: An open framework for openflow switch evaluation. In *International Conference on Passive and Active Network Measurement*, pages 85–95. Springer, 2012.

[32] Project floodlight SDN controller. `https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview`. [Accessed March 2020].

[33] M Rasih Celenlioglu and H Ali Mantar. An SDN based intra-domain routing and resource management model. In *International Conference on Cloud Engineering*, pages 347–352. IEEE, 2015.

[34] Jinyao Yan, Hailong Zhang, Qianjun Shuai, Bo Liu, and Xiao Guo. Hiqos: An SDN-based multipath QoS solution. *China Communications*, 12(5):123–133, 2015.

[35] Sana Tariq and Mostafa Bassiouni. Qamo-sdn: QoS aware multipath tcp for software defined optical networks. In *Annual Consumer Communications and Networking Conference (CCNC)*, pages 485–491. IEEE, 2015.

[36] Che Huang, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida. A multipath controller for accelerating gridftp transfer over SDN. In *International conference on e-science*, pages 439–447. IEEE, 2015.

[37] Syed Asad Hussain, Shuja Akbar, and Imran Raza. A dynamic multipath scheduling protocol (dmsp) for full performance isolation of links in software defined networking (SDN). In *Workshop on Recent Trends in Telecommunications Research (RTTR)*, pages 1–5. IEEE, 2017.

[38] Atef Abdelkefi and Yuming Jiang. A structural analysis of network delay. In *2011 Ninth Annual Communication Networks and Services Research Conference*, pages 41–48. IEEE, 2011.

[39] Zhenhong Shao, Yongxiang Liu, Yi Wu, and Lianfeng Shen. A rapid and reliable disaster emergency mobile communication system via aerial ad hoc bs networks. In *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pages 1–4. IEEE, 2011.

[40] Mathieu Dervin, Isabelle Buret, and Céline Loisel. Easy-to-deploy emergency communication system based on a transparent telecommunication satellite. In *Proceedings of the International Conference on Advances in Satellite and Space Communications (SPACOMM)*, pages 168–173. IEEE, 2009.

[41] Abobakr Y Shahrah and Majed A Al-Mashari. Emergency response systems: research directions and current challenges. In *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, page 161. ACM, 2017.

[42] Avgoustinos Filippoupolitis and Erol Gelenbe. A decision support system for disaster management in buildings. In *Computer Simulation Conference*, pages 141–147. Society for Modeling & Simulation International, 2009.

[43] Kaiqi Xiong, Kyoung-Don Kang, and Xiao Chen. A priority-type resource allocation approach in cluster computing. In *Proceedings of the International Conference on High Performance Computing and Communications (HPCC)*, pages 271–278. IEEE, 2011.

[44] Kaiqi Xiong and Harry Perros. SLA-based service composition in enterprise computing. In *IWQoS*, pages 30–39. IEEE, 2008.

[45] Vita Lanfranchi and Neil Ireson. User requirements for a collective intelligence emergency response system. In *the British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*, pages 198–203. British Computer Society, 2009.

[46] Keshav Sood, *et al.* Are current resources in SDN allocated to maximum performance and minimize costs and maintaining QoS problems? In *Proceedings of the Australasian Computer Science Week Multiconference*, page 42. ACM, 2017.

[47] Sushant Jain, ,, *et al.* B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[48] Murat Karakus and Arjan Durresi. Quality of service (qos) in software defined networking (SDN): A survey. *Journal of Network and Computer Applications*, 80:200–218, 2017.

[49] Shuo Wang, *et al.* Flowtrace: measuring round-trip time and tracing path in software-defined networking with low communication overhead. *Frontiers of Information Technology & Electronic Engineering*, 18(2):206–219, 2017.

[50] Wang Miao, *et al.* Performance modelling and analysis of software-defined networking under bursty multimedia traffic. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 12(5s):1–19, 2016.

[51] Siamak Azodolmolky, *et al.* An analytical model for software defined networking: A network calculus-based approach. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 1397–1402. IEEE, 2013.

[52] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. A survey of security in software defined networks. *Communications Surveys Tutorials*, 18(1):623–654, Firstquarter IEEE, 2016.

[53] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 1629–1637. IEEE, 2011.

[54] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the Conference on Computer and Communications Security (CCS)*, pages 413–424. ACM, 2013.

[55] Yun Tian, Vincent Tran, and Mutalifu Kuerban. DoS attack mitigation strategies on SDN controller. In *Proceedings of the Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0701–0707. IEEE, 2019.

[56] Guowei Wu, Zhaoxin Li, and Lin Yao. DoS mitigation mechanism based on non-cooperative repeated game for SDN. In *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*, pages 612–619. IEEE, 2018.

[57] Gao Shang, Peng Zhe, Xiao Bin, Hu Aiqun, and Ren Kui. Flooddefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2017.

[58] Pengpeng Wu, Lin Yao, Chi Lin, Guowei Wu, and Mohammad S Obaidat. Fmd: A DoS mitigation scheme based on flow migration in software-defined networking. *International Journal of Communication Systems*, 31(9):e3543, Wiley Online Library, 2018.

[59] Tao Wang, Hongchang Chen, and Chao Qi. Mindos: A priority-based SDN safe-guard architecture for DoS attacks. *IEICE*, 101(10):2458–2464, 2018.

[60] NA Bharathi, V Vetriselvi, and Ranjani Parthasarathi. Mitigation of DoS in SDN using path randomization. In *Proceedings of the International Conference on Computer Networks and Communication Technologies (ICCNCT)*, pages 229–239. Springer, 2019.

[61] Tao Wang, Hongchang Chen, Guozhen Cheng, and Yulin Lu. Sdnmanager: A safeguard architecture for SDN DoS attacks based on bandwidth prediction. *Security and Communication Networks*, 2018, 2018.

[62] Song Wang, Karina Gomez Chavez, and Sithamparanathan Kandeepan. Seco: SDN secure controller algorithm for detecting and defending denial of service attacks. In *Proceedings of the International Conference on Information and Communication Technology (ICoIC)*, pages 1–6. IEEE, 2017.

[63] Song Wang, Sathyanarayanan Chandrasekharan, Karina Gomez, Sithamparanathan Kandeepan, Akram Al-Hourani, Muhammad Rizwan Asghar, Giovanni Russello, and Paul Zanna. Secod: SDN secure control and data plane algorithm for detecting and defending against DoS attacks. In *Proceedings of the Network Operations and Management Symposium (NOMS)*, pages 1–5. IEEE, 2018.

[64] Jing Zheng, Qi Li, Guofei Gu, Jiahao Cao, David KY Yau, and Jianping Wu. Realtime ddos defense using cots SDN switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security*, 13(7):1838–1853, 2018.

[65] Shuhua Deng, Xing Gao, Zebin Lu, Zhengfa Li, and Xieping Gao. DoS vulnerabilities and mitigation strategies in software-defined networks. *Journal of Network and Computer Applications, Elsevier*, 125:209–219, 2019.

[66] Mutalifu Kuerban, Yun Tian, Qing Yang, Yafei Jia, Brandon Huebert, and David Poss. Flowsec: DoS attack mitigation strategy on SDN controller. In *Proceedings of the International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–2. IEEE, 2016.

[67] Vern Paxson. Zeek: The Zeek Network Security Monitor. *Computer Networks*, 31(23-24):2435–2463, 1999.

[68] Martin Roesch, *et al.* Snort-lightweight intrusion detection for networks. In *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, volume 99, pages 229–238, 1999.

[69] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop*, 2003.

[70] Jiri Navratil and R Les Cottrell. Abwe: A practical approach to available bandwidth estimation. In *Proc. passive and active measurement workshop*, 2003.

[71] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[72] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. GENi: A federated testbed for innovative network experiments. *Computer Networks*, 61:5 – 23, 2014. Special issue on Future Internet Testbeds – Part I.

[73] Hong Chen and David D Yao. *Fundamentals of queueing networks: Performance, asymptotics, and optimization*, volume 46. Springer Science & Business Media, 2013.

[74] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, (4):397–413, 1993.

[75] Ivan Delchev. Linux traffic control. In *Networks and Distributed Systems Seminar, International University Bremen, Spring*, 2006.

[76] Rajendra K Jain, Dah-Ming W Chiu, and William R Hawe. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[77] Prashant Kumar, Meenakshi Tripathi, Ajay Nehra, Mauro Conti, and Chhagan Lal. Safety: Early detection and mitigation of tcp syn flood utilizing entropy in SDN. *Transactions on Network and Service Management (TNSM)*, 15(4):1545–1559, IEEE, 2018.

[78] Jon Postel. Internet control message protocol. 1981.

Appendix A: Glossary

AIMD	Additive Increase Multiplicative Decrease
API	Application Programming Interface
AS	Autonomous System
CDF	Cumulative Distribution Function
DC	Downlink Channel
DDoS	Distributed Denial of Service
D-ITG	Distributed Internet Traffic Generator
DoS	Denial of Service
DPI	Deep Packet Inspection
ERS	Emergency Response System
EWMA	Exponentially Weighted Moving Average
FCFS	First Come First Served
FSC	Flow Statistics Collection
GENI	Global Environment for Networking Innovations
ICT	Information and Communications Technology
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IoT	Internet of Things

ISP	Internet Service Provider
IXP	Internet Exchange Points
LDS	Link Discovery Service
LLDP	Link Layer Discovery Protocol
MITM	Man-in-the-Middle
MPTCP	Multi-Path-TCP
NOS	Network Operating System
OVS	Open vSwitch
QoS	Quality of Service
RED	Random Early Detection
REST	Representational State Transfer
RTT	Round Trip Time
Rx	Receive Speed
SDN	Software-Defined Networking
Science DMZ	Science De-Militarized Zone
SLA	Service Level Agreement
SP	Service Provider
SPAN	Switch Port Analyzer
SS	Switch Server
TCB	Transmission Control Block
TE	Traffic Engineering
TS	Time Series
Tx	Transmit Speed
UC	Uplink Channel
WMA	Weighted Moving Average
XMPP	Extensible Messaging and Presence Protocol