Prototyping an SDN Control Framework for QoS Guarantees

Mohamed Rahouti[1(✉)], Kaiqi Xiong[2], and Yufeng Xin[3]

[1] Fordham University, The Bronx, NY 10458, USA
mrahouti@fordham.edu
[2] University of South Florida, Tampa, FL 33620, USA
xiongk@usf.edu
[3] University of North Carolina, Chapel Hill, NC 27599, USA
yxin@renci.org

Abstract. The centralized control capability of Software-Defined Networking (SDN) gives us an excellent opportunity to enhance the Quality of Service (QoS) routing. The end-to-end QoS-aware traffic forwarding must consider the computation latency associated with optimal path selection while reducing the controller's response time. In this paper, we propose a new SDN controller framework that consists of a queueing mechanism, active link delay measurements, efficient statistic estimate of network states, and intelligent path computation and selection. We implement our framework as a modular application in a Floodlight SDN controller software and conduct comprehensive experimental studies on the Global Environment for Network Innovations (GENI) testbed. Our performance evaluation based on experimental results demonstrates that the proposed framework can significantly reduce the latency in both the control plane and data plane, and find optimal paths with the minimum end-to-end delay.

Keywords: Software-Defined Networking · OpenFlow · Global environment for networking innovations · Path selection · Latency · Queueing · Quality of Service

1 Introduction

Software-Defined Networking (SDN) has become a vital network paradigm for expediting network management and control. It provides a global view of network resources and separates its control layer from its infrastructure layer [5]. Via a logically centralized controller, an SDN permits programmable interfaces to insert and push forwarding rules into forwarding devices (e.g., Open vSwitch) flow table for efficient network communication [14]. In order to better leverage SDN capabilities, the delicate holistic view of flows must be regarded as contributory to networking applications (e.g., traffic engineering [5], QoS-aware flow routing [15,16], and flow inspection [6]).

Y. Weng et al. (Eds.): TridentCom 2020, LNICST 380, pp. 3–16, 2021.
https://doi.org/10.1007/978-3-030-77428-8_1

Various research studies have been carried out to explore and enhance the Quality of Service (QoS) in SDN environments. The majority of previous studies have shed light on the data plane latency performance in terms of packet trip time. The end-to-end latency involves multiple factors, including propagation delay, switching latency, and control plane latency. It is important to note that the data-control communication delay (i.e., the delay associated with sending packets from a switch to the controller and back/OpenFlow packet_ins and packet_outs) plays a significant role in the end-to-end delay of traffic delivery. Therefore, the centralized control of networking traffic in SDN networks presents an additional factor that can not be ignored in designing latency control mechanisms.

To ensure QoS, a typical solution can be the dedication of more resources, such that the controller's response time is reduced. However, such a solution may significantly increase the complexity and operational cost of the infrastructure (e.g., energy and consumption cost). Moreover, the processing resources of SDN controllers at the service provider (SP) can be limited (i.e., SP restricts the concurrent virtual machine instances per account), where application-specific QoS may be adversely degraded. Thus, it is essential to note that the stability of QoS is crucial as it dominates the cost of infrastructural operation. Hence, aspects such as the controller response time (i.e., latency), flow rule processing, and forwarding delay have mutual dependencies, subsequently impacting the end-to-end communication delay and provisioned QoS. Therefore, it is of great interest to elaborate sophisticated processing and forwarding strategies at both the control and data planes to maintain QoS and optimize the operational cost without affecting the controller overheads (e.g., response time).

In this work, we address the end-to-end latency optimization problem of OpenFlow networks based on a systematic enhancement on both the control and data planes. Specifically, we leverage a prioritized service queueing model along with active measurements of the per-link latency mechanism to achieve optimization of per-flow end-to-end latency. Namely, the QoS mechanism we present is based on real-time per-link latency estimation and selects forwarding paths with minimal delay based upon an adaptive heuristic approach.

We implemented the proposed mechanisms in the Floodlight controller software. We then carried out comprehensive experiments in a live SDN network built in the GENI testbed to validate the proposed solution's efficiency. The experimental results demonstrate that our design model guarantees a significant improvement in end-to-end delay and controller response times.

Our contributions in this study are outlined as follows.

- Integration of controller's statistics into the design of a real-time end-to-end latency metric estimation and path selection.
- Adaptation of a queueing scheme to optimize controller's response time.
- Prototype and validation of the proposed approach in the multiple sites GENI testbed.

The rest of this paper is organized as follows. In Sect. 2, we further summarize existing studies and related work. Building upon this, in Sect. 3, we present a QoS

model and respective architectural design of the SDN framework to resolve our research problem. In Sect. 4, we table validation methodologies of our proposed solution along with corresponding experimental findings. Finally, we conclude the paper discussion with future work in Sect. 5.

2 Related Work

In networking-enabled environments, QoS is regarded as the qualitative mensuration of networking services' overall performance, such as end-to-end communication delay [5]. Carrying out reliable QoS in SDN networks is of an essential benefit to providing dependable and efficient communication services and networking applications [17]. Thus, many recent studies have addressed the performance of OpenFlow devices. Yet, they have not provided effective solutions aligning in tandem with applications that require stringent optimization at both the control and data planes.

Most notably, some studies such as Huang et al. [10] and Rotsos et al. [19] studied the performance of SDN switching over various OpenFlow device vendors, whereas Curtis et al. [7] presented a notable mechanism to improve controller response times and switching delays. Over and above these studies, others tried to improve QoS guarantees in SDN networks using different approaches. Namely, Egilmez et al. [8] proposed an OpenFlow mechanism by leveraging the end-to-end bandwidth allocation for multimedia streaming, whereas Sharma et al. [20] and Celenlioglu and Mantar [4] presented SDN frameworks by leveraging open source controller software. The solution presented in [4] also tried to improve routing scalability and resource management, which subsequently enhances QoS guarantees in stationary nodes.

Other studies such as Jinyao et al. [22], Tariq and Bassiouni [21], and Hussain et al. [11] also tried to improve end-to-end communication QoS in SDN networks. Specifically, Jinyao et al. [22] leveraged queuing schemes for managing resource and bandwidth allocation along with an adjusted version of the Dijkstra algorithm to minimize delays in traffic delivery. Similarly, Tariq and Bassiouni [21] a multi-path-TCP (MPTCP) solution (for simulated SDN-enabled data centers) based upon best paths pre-computation between end-user nodes. Lastly, Hussain et al. [11] assessed a mechanism for hashed multi-path search in Floodlight controller software [1], where traffic scheduling and forwarding is performed via a hash function flow balancing service. Now since SDN controllers provide global traffic monitoring capabilities, various traffic engineering (TE) implementations also deploy these capabilities to enhance communication QoS. Namely, Google interconnected data centers (using SDN) while balancing capacity utilization of networks based upon application-specific priority [12]. Agarwal et al. [3] also tried to improve traffic delays and packet losses in environments interconnecting both OpenFlow and traditional routing devices. Furthermore, Jarschel et al. [13] tabled application-aware traffic management using Deep Packet Inspection (DPI) techniques. Further studies such as [23] resolved multimedia streaming QoS based upon a multi-priority flow scheme, such that high priority traffic is temporarily rerouted following congestion occurrence.

It is important to note that many past studies focused on TE in OpenFlow networks for QoS-enabled queueing tried to identify data plane traffic by port number with low accuracy (or even simply assume network can classify incoming traffic). Therefore, these solutions become impractical for real-world SDN environments. Furthermore, the majority of these summarized solutions have been tested over simulated experiments with impractical QoS criteria. To the best of our knowledge, none of these studies have proposed a practical SDN-enabled framework based upon combining and integrating a priority-based data-control queueing mechanism along with TE techniques to optimize the control response time and end-to-end traffic delay.

Nevertheless, despite these contributions, various open challenges remain here. For example, there is a further need to leverage the real-time data collection/processing capabilities and queueing mechanisms (at both the controller and OpenFlow switch) to achieve more effective traffic forwarding and dynamic path computation. This is indeed 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 adequately integrated with SDN control plane traffic. Building upon the framework presented in [18], we propose an extended prototyping design with enhanced mechanisms to better improve the end-to-end QoS for traffic delivery with optimized controller's response time. Consider the design details presented next.

3 Design and Implementation

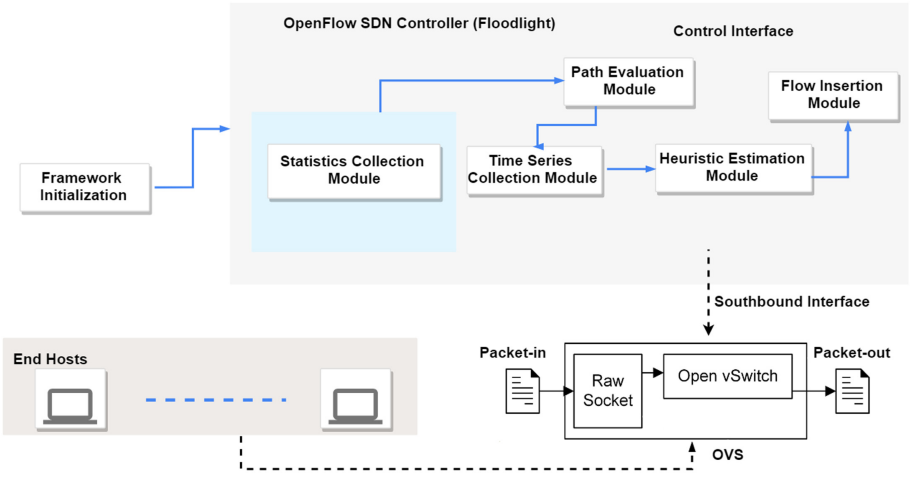

Fig. 1. Architectural design of the proposed framework.

In this work, the end-to-end communication scenario is contemplated in an SDN network, and the end nodes send and receive flows at a fixed rate across the networking environment. We are specifically concerned about communication delays, where the switch-controller communication (i.e., switch to controller channel), topology statistics calculation, and paths update are contributing factors in the packet forwarding delays [24]. In SDN delay-centric applications (i.e., applications that require low latency), an efficient flow forwarding mechanism must try to optimize the average end-to-end delay. Therefore, providing a close-to-optimal end-to-end delay for these applications is of great importance here to ensure efficient usage of link capacity and resources [5]. We develop an SDN mechanism by leveraging specific queueing mechanisms and traffic engineering techniques to address the presented challenges. Consider the details next.

The proposed framework architecture/modules are depicted in Fig. 1 and consists of four main modules, traffic classifier (queuing scheme), statistics collector, time series estimator, and optimal path selection. Modules are described and detailed in the following subsections.

3.1 Traffic Classifier and Queue Assignment

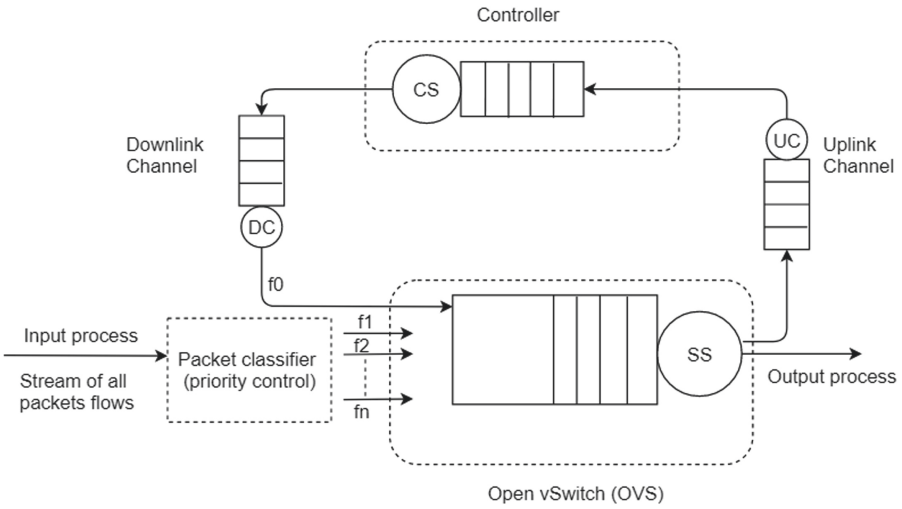

Fig. 2. The prioritized service-enabled queueing scheme for a data-control communication channel.

Many researchers have modeled the data plane as a Jackson network while the SDN controller as a finite queue. The shortcoming in such a model is that the model treats both control flows and data plane flows with an equal priority (i.e., normal networking traffic). According to recent studies, unlike SDN controllers, OpenFlow switches provide a reliable processing capacity of packets and forward

packets in a microsecond. Moreover, an SDN controller is likely not to completely process *packet_in* messages with a very high arrival rate even when the processing time of a *packet_in* message may be few milliseconds (e.g., the arrival rate is bigger than thousands of packets per second).

The design of our proposed framework further relies upon an integrated set of queues. Namely, the uplink channel (UC) and downlink channel (DC) queues for switch-to-control and control-to-switch control packets buffering. Such queues are assumed to be of an infinite length. Moreover, the controller consists of a finite length queue to buffer arriving Packet_Ins/lookup requests. Meanwhile, the SDN-enabled switch implements a set of priority-based queues for incoming user flows buffering (after classification). Specifically, four priority levels are enabled here, i.e., normal, low, medium, and high. The control plane queues (DC, UC, and controller) are designed for management and control of flow rules, whereas the respective data plane queues buffer data packets with different priority levels. Finally, it is important to note that our framework's control packets are prioritized over data plane flows regardless of the latter's priority level. Therefore, throughout packet scheduling, control plane flows (i.e., in the control channel queue) are ensured a minimal delay of processing. In contrast, the data plane flows are handled according to their arrival priority levels.

Furthermore, when a new packet arrives in the switch, the switch-controller communication channel causes more latency towards the end-to-end packet delay. Therefore, a not-matched packet with existing flow rules can affect the rigorous end-to-end delay bound more than matched packets. Thus, a prioritized service queueing scheme is implemented along with our SDN framework, as depicted in Fig. 2. A priority queue is implemented in the infrastructure layer to separate flows that match existing flow rules (i.e., a routing path already calculated) from new packet arrivals. Moreover, the classification module classifies incoming data packets (client flows) into multiple priority levels and then buffers them separately. Note that a feedback control signal is also utilized to design the efficient priority-based transmission of all enqueued data flows, i.e., with the SDN controller performing dynamic queue control, detailed in a later section.

As the OpenFlow standards do not provide specifications to assign and configure queues, our OpenFlow switch is set up to isolate and classify traffic by enqueuing them on a separate queue on the egress port. The assigned queue on the data plane has an associated QoS configuration, including the queue capacity and the enqueued traffic service rate.

The purpose behind implementing such a queueing scheme is to minimize the overall traffic delay (between a given pair of nodes, i and j), which is partially impacted by the control plane latency given in the following equation (assuming the incoming packet cannot be matched by flow rules in the respective switch).

$$L^{n_i}_{prop} + \sum_{k=1}^{w+1} L^{m}_{trans} + \sum_{k=1}^{w} (L^{m}_{queue} + L^{m}_{proc}) + Ctrl_{i,j}, \tag{1}$$

where $L^{n_i}_{prop}$ is the propagation delay at host n_i, L^{m}_{trans} is the transmission delay at the m^{th} link, L^{m}_{queue} is the queuing delay at the m^{th} switch, and L^{m}_{proc}

is the processing delay at the m^{th} switch. Moreover, $ctrl_{i,j}$ is the delay of the control path due to the switch-controller communication (i.e., flow rule querying and insertion). Based on (1), the data-control channel latency can be induced by subtracting the respective measured delays from the overall end-to-end delay as it will be shown in Sect. 4.

Lastly, it is essential to note that the OpenFlow software version 1.5 (used in this work) standard offers several QoS capabilities and actions [2]. Specifically, the $set-queue$ features can associate a given queue ID for data flows, which can then be forwarded to further ports based upon a specific output function. Based upon the queue ID, we can decide which queue is attached to which port to schedule and or forward the data flow. However, these integral QoS-enabled queueing mechanisms and actions only provide basic QoS support, and thus further enhancements and implementations remain of substantial consideration. Therefore, in this work, we build upon the basic integral QoS capabilities in OpenFlow software to implement more complex QoS policing functions to classify data flows in multiple queue categories (while prioritizing control flows).

3.2 Latency Collection

The per-link latency is calculated based upon timestamps injections by the SDN controller via the Link Layer Discovery Protocol (LLDP) packets. Specifically, the SDN controller transmits these LLDP packets (i.e., as packet-outs) to all enabled switches in the network. Once the controller receives one of its transmitted LLDPs, it processes it as a Packet_In. Building upon this, the controller examines the LLDP packet timestamp and subtracts it from its current time (i.e., this computation gives the elapsed time). Lastly, the per-link latency will be the computation result of the elapsed time minus the latencies associated with switch-to-controller for the origin switch and the one that sent the Packet_In. For simplicity, the LLDP-based latency computation in our framework is induced as follows.

$$t\_lldp\_tx - t\_lldp\_rx - lat\_tx\_ovs - lat\_rx\_ovs \tag{2}$$

where t_lldp_tx, t_lldp_rx, lat_tx_ovs, and lat_rx_ovs present the initial timestamp of the LLDP packet, the round trip time, latency in a neighboring switch that first receives and sends out the LLDP packet to the next switch, and latency in next-hop switch, respectively.

3.3 Latency Estimation

To minimize the controller's overhead caused by statistics collection, we implement an exponentially weighted moving average (EWMA) model for the short-run per-link latency metric estimation. This module uses per-link latencies calculated by the previous module (discussed in Subsect. 3.2), and obtain a new estimation of accumulated per-link latencies for an initialized window size that must be set by the network operator before launching the framework (10 sec. by default).

3.4 Path Selection

Inspired by the $A*$ algorithm [9], we implement a search heuristic $h(n)$ (where n is the source node) in the path selection module for the heuristic search of the optimal path from a source to a target node. Our path selection module only utilizes readily acquired per-link latency metric estimations (i.e., outputs of the latency estimation module described in Subsect. 3.3) between a particular communicating pair of nodes (source and target). The heuristic $h(n)$ is regarded as a set of paths between two communicating nodes, where the path cost is an estimate rather than an actual. In our path selector module implementation, we maintain two lists, $OPEN$ and $CLOSED$, for the list of pending paths (i.e., entities that are already visited but not expanded, and thus their successors are not checked yet). The list of entities that have been checked/expanded, and their successor nodes have been identified and put in the $OPEN$ list, respectively.

Building upon this, once the path estimates are obtained from the previous module, the path selection module uses them to compute the optimal path with minimal end-to-end latency. We select only the best two paths for overhead minimization if there are multiple paths between source and target nodes.

3.5 Queue Control and Control Flows Prioritization

In order for us to control the queue length and avert congestion, a minimum and maximum thresholds are defined and used for maintaining the average queue size at a midway (i.e., assuming the level of network congestion is medium). Furthermore, the proposed queue control mechanism averts dropping high priority packets. This aversion is achieved through the real-time computation of average queue size per data plane queue. The queue averages are updated based upon the controller feedback control message, i.e., if one or more data plane queues are congested, the forwarding device (OVS) is instructed to drop lower priority packets. Finally, to assure control packets are properly prioritized over data flows, the OpenFlow *Pause* action/message is leveraged. According to its flow table, the OVS can pause a packet's forwarding procedure while serializing the current packet state as *continuation* in the *Packet_In* message). Later on, the controller transmits a continuation flag back to the OVS using NXT_RESUME action, i.e., to resume flow processing from its previous interruption point.

4 Evaluation

In this section, we discuss the construction of our experimental topology on the Global Environment for Network Innovations (GENI), hardware and software setups, and experimental findings for our proposed framework's performance.

4.1 GENI Topology

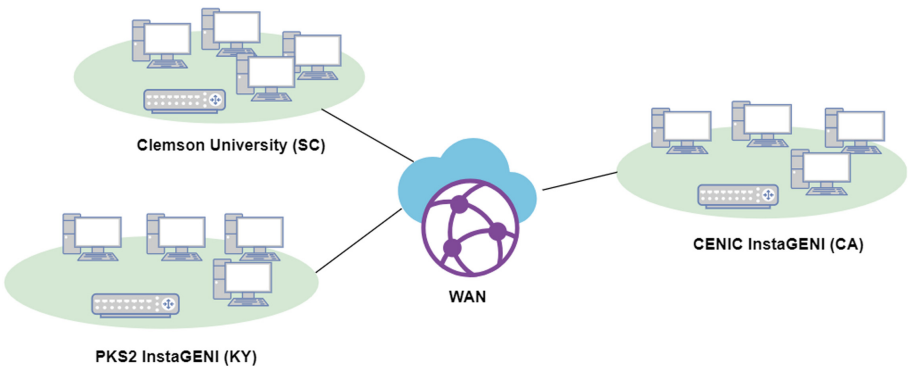

Fig. 3. Stitching multiple sites through GENI testbed for inter-region communications. The wide-area communication at scale is not restricted to one geographical site. GENI testbed allows inter-network domains for inter-region communications to support the elaboration of real-world experiments over the public Internet.

As the at-scale wide-area communication is not limited to one geographical region, GENI testbed grants inter network domains for inter-region communication in order to assist with elaborating real-world experiments over public Internet traffic. Thus, to better evaluate our framework's efficiency and evaluate the end-to-end delay within a topology connecting multiple sites, we deployed networking resources from multiple sites through the stitching capability of GENI, specifically, CA, KY, and SC as shown in Fig. 3. The constructed topology is depicted in Fig. 4. The GENI testbed's stitching capability allows for connecting nodes provided by multiple aggregates into a coherent real-world network.

4.2 Resource Setup

In order for us to conduct a realistic evaluation of the proposed SDN framework, the constructed networking topology must consist of the following.

1. SDN switching and routing devices
2. Software module to emulate networking traffic
3. Multiple paths and links with changeable throughput and packet loss

Additionally, open-source SDN controller and OpenFlow software implementations are chosen to ensure broader 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 network. This well-established testbed allows

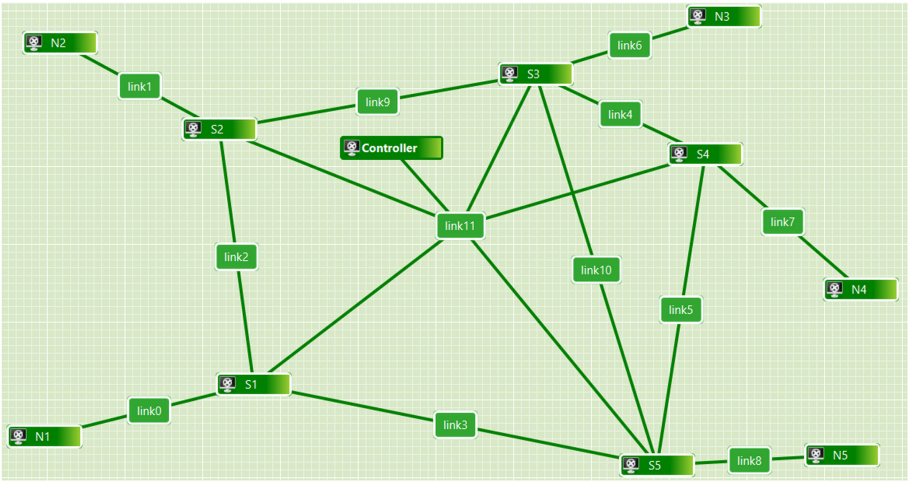

Fig. 4. A Constructed topology on GENI testbed, where multiple geographical locations (sites) are inter-connected through the stitching capabilities of GENI

researchers to build arbitrary network topology slices and deploy tailored end hosts (traffic generation) and SDN controllers.

Specifically, networking communications in these experiments are performed using mainly two tools, iPerf and hping3, as normal and burst traffic, respectively. Additionally, our at-scale topology's architectural design utilizes Open vSwitch (OVS) that supports OpenFlow software 1.5 along with Floodlight 1.2 as an SDN controller software, where the data plane queue (switching queue) is initially configured as follows.

```
$ ovs-vsctl add-br s0
$ ovs-vsctl add-port s0 eth0
$ ovs-vsctl set port eth0
qos=@newqos -- --id=@newqos create qos
type=linux-htb
other-config:max-rate=5000000
queues:0=@newqueue --
--id=@newqueue create queue
other-config:min-rate=3000000
other-config:max-rate=3000000
```

4.3 Performance Results

The reported experimental runs are conducted for 15 successive trials in an automated manner, and the results are then averaged. Detailed results are now presented for % improvement of end-to-end delay, comparisons of end-to-end delay, and data-to-control latency between our framework and the default SDN controller software.

Table 1. % improvement of end-to-end delay in our framework with contrast to the default SDN Floodlight.

Statistic	Node N_1	Node N_2	Node N_3
Mean	6.02%	8.14%	9.38%
Median	6.76%	9.73%	8.92%

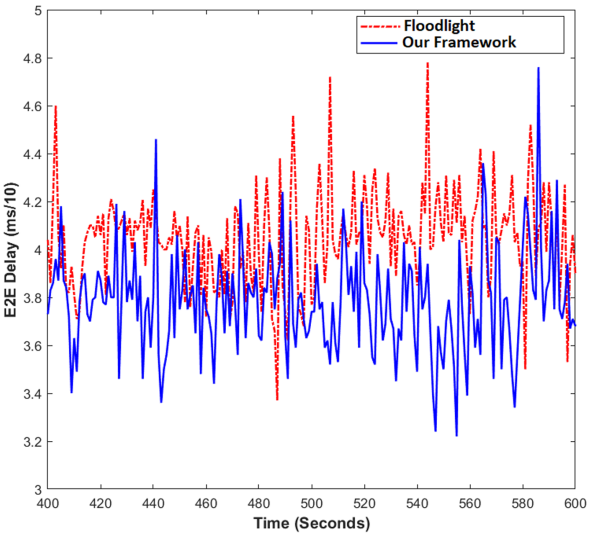

Fig. 5. A comparison of end-to-end delay between our framework and the default Floodlight SDN controller

Foremost, Our evaluation findings demonstrated that our developed framework outperforms the default optimization configurations in Floodlight SDN controller software. Table 1 and Fig. 5 give a summary of our comprehensive experiments. Specifically, Table 1 gives % end-to-end delay improvement in our framework, while Fig. 5 depicts a visual presentation of the delay in an end-to-end communication between designated nodes in our topology. In these experiments, we point N_1 and N_6 as a source node and a destination node, respectively. Figure 5 demonstrates our system outperforms the path selection mechanism adopted by Floodlight, and implies smaller end-to-end traffic delay. Table 1 shows that the proposed solution improves the average end-to-end delay by about 10%. This delay percentage is vital when delay-sensitive applications (e.g., emergency response systems) are built upon networked systems.

In this experiment, we aim at examining the efficiency of our solution in terms of control plane latency optimization between the same pair of switches in our GENI topology. Herein, the experiments take place on the same path between the switching devices. Moreover, it is important to state that we introduce additional networking traffic using the iPerf tool to simulate and perform link congestion at

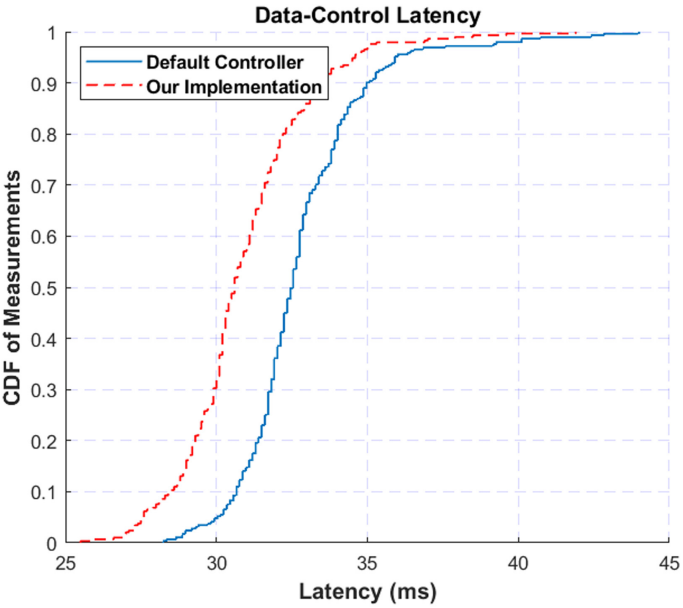

Fig. 6. A comparison of data-to-control latency in our framework vs. the one provided by the default Floodlight SDN controller software with a burst of traffic.

the intermediate switch on the path by adjusting the traffic burst. Specifically, we communicate 200 Mbps iPerf traffic shaping in this experimental scenario. We run both our framework and the non-optimized SDN controller (default Floodlight controller software) separately for five minutes.

Based on (1), we can induce the data-control channel latency by subtracting the respective measured delays from the overall end-to-end delay. Figure 6 presents the cumulative distribution functions (CDFs) of control-data plane latencies provided by our framework and default Floodlight SDN controller. As shown in Fig. 6, our queueing scheme outperforms the default controller with regard to control plane latency optimization.

5 Conclusions

In this paper, we presented a new SDN controller framework that enhances end-to-end QoS awareness. The proposed system utilizes various traffic engineering (TE) techniques, a queueing mechanism, real-time latency estimation, and heuristic path search to optimize the end-to-end latency in traffic delivery. The system enables prioritization of control plane traffic while minimizing the control plane latency through a priority queueing scheme. Our comprehensive experimentation was carried out over a continent-scale stitched topology (over multiple sites) built on the Global Environment for Network Innovations (GENI), a heterogeneous real-world testbed to experimentally evaluate the efficiency of

the proposed framework. In the future work, we plan to integrate this framework into multi-domain SDN environment and explore other QoS metrics such as classification of networking traffic of varying priority levels and flow routing and forwarding with respect to the identified priorities.

Acknowledgment. We would like to acknowledge the National Science Foundation (NSF) that partially sponsored the work under grants #1633978, #1620871, #1620862, #1651280, #1531099 and BBN/GPO project #1936 through NSF/CNS grant. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of NSF.

References

1. Floodlight openflow controller project. http://www.projectfloodlight.org/floodlight/
2. Openflow switch specification, version 1.5.1 (protocol version 0x06). https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
3. Agarwal, S., Kodialam, M., Lakshman, T.: Traffic engineering in software defined networks. In: International Conference on Computer Communications, pp. 2211–2219. IEEE (2013)
4. Celenlioglu, M.R., Mantar, H.A.: An SDN based intra-domain routing and resource management model. In: International Conference on Cloud Engineering, pp. 347–352. IEEE (2015)
5. Chin, T., Rahouti, M., Xiong, K.: Applying software-defined networking to minimize the end-to-end delay of network services. ACM SIGAPP Appl. Comput. Rev. **18**(1), 30–40 (2018)
6. Chin, T., Xiong, K., Rahouti, M.: SDN-based kernel modular countermeasure for intrusion detection. In: Lin, X., Ghorbani, A., Ren, K., Zhu, S., Zhang, A. (eds.) SecureComm 2017. LNICST, vol. 238, pp. 270–290. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78813-5_14
7. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: DevoFlow: scaling flow management for high-performance networks. In: SIGCOMM Computer Communication Review, vol. 41, pp. 254–265. ACM (2011)
8. Egilmez, H.E., Dane, S.T., Bagci, K.T., Tekalp, A.M.: OpenQoS: an openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In: Asia Pacific Signal and Information Processing Association Annual Summit and Conference, pp. 1–8. IEEE (2012)
9. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968). https://doi.org/10.1109/tssc.1968.300136
10. Huang, D.Y., Yocum, K., Snoeren, A.C.: High-fidelity switch models for software-defined network emulation. In: SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 43–48. ACM (2013)
11. Hussain, S.A., Akbar, S., Raza, I.: 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, pp. 1–5. IEEE (2017)

12. Jain, S., et al.: B4: experience with a globally-deployed software defined WAN. In: SIGCOMM Computer Communication Review, vol. 43, pp. 3–14. ACM (2013)
13. Jarschel, M., Wamser, F., Hohn, T., Zinner, T., Tran-Gia, P.: SDN-based application-aware networking on the example of Youtube video streaming. In: European Workshop on Software Defined Networks, pp. 87–92. IEEE (2013)
14. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
15. Ni, H., Rahouti, M., Chakrabortty, A., Xiong, K., Xin, Y.: A distributed cloud-based wide-area controller with SDN-enabled delay optimization. In: Power & Energy Society General Meeting, pp. 1–5. IEEE (2018)
16. Rahouti, M., Xiong, K., Chin, T., Hu, P.: 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, pp. 18–23. IEEE (2018)
17. Rahouti, M., Xiong, K., Chin, T., Hu, P., De Oliveira, D.: A preemption-based timely software defined networking framework for emergency response traffic delivery. In: International Conference on High Performance Computing and Communications; International Conference on Smart City; International Conference on Data Science and Systems, pp. 452–459. IEEE (2019)
18. Rahouti, M., Xiong, K., Xin, Y., Ghani, N.: Latencysmasher: a software-defined networking-based framework for end-to-end latency optimization. In: 2019 IEEE 44th Conference on Local Computer Networks, pp. 202–209. IEEE (2019)
19. Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., Moore, A.W.: OFLOPS: an open framework for OpenFlow switch evaluation. In: Taft, N., Ricciato, F. (eds.) PAM 2012. LNCS, vol. 7192, pp. 85–95. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28537-0_9
20. Sharma, S., et al.: Implementing quality of service for the software defined networking enabled future Internet. In: The European Workshop on Software Defined Networking, pp. 49–54. IEEE (2014)
21. Tariq, S., Bassiouni, M.: QAMO-SDN: QoS aware multipath TCP for software defined optical networks. In: Annual Consumer Communications and Networking Conference, pp. 485–491. IEEE (2015)
22. Yan, J., Zhang, H., Shuai, Q., Liu, B., Guo, X.: HiQoS: an SDN-based multipath QoS solution. China Commun. **12**(5), 123–133 (2015)
23. Yu, T.F., Wang, K., Hsu, Y.H.: Adaptive routing for video streaming with QoS support over SDN networks. In: International Conference on Information Networking, pp. 318–323. IEEE (2015)
24. Zhang, T., Liu, B.: Exposing end-to-end delay in software-defined networking. Int. J. Reconfig. Comput. **2019** (2019)