Journal of Applied Informatics and Computing (JAIC)
Vol.10, No.4, August 2026, pp. 3140~3147
e-ISSN: 2548-6861

A Lightweight Qdisc Triggered Controller (QTC) for Congestion Aware Software Defined Networks

Mustafa Ayad Anwer*[1], Zaid Shakir Kadhim Al-Shammari*[2], Ahmed Sabri Ghazi Behadili*[3]
*College of Engineering, Al-Karkh University of Science, Baghdad, 10081, Iraq.
mustafa.alani@kus.edu.iq [1], zaid.shaker.elc@kus.edu.iq [2], eng.ahmed.sabri@kus.edu.iq [3]

Article Info

Article history:

Received 2026-07-09
Revised 2026-07-24
Accepted 2026-07-31

Keyword:

Software Defined Networking (SDN),
Qdisc triggered controller,
Mininet,
Ryu,
Congestion aware,
Adaptive path switching,
OpenFlow.

ABSTRACT

Software Defined Networking (SDN) enables centralized and programmable traffic management. However, maintaining service quality under congestion remains a challenging task while using heavy computation or delayed network wide optimization. This paper presents a lightweight Qdisc Triggered Controller (QTC) for congestion aware path switching in SDN. The suggested controller monitors queueing discipline statistics specifically packet drops and qdisc overlimit events. The proposed mechanism is implemented using Ryu controller and evaluated in a Mininet based topology under normal, persistent, transient and sequential bottleneck scenarios. The experimental findings show that QTC maintains near baseline performance under normal network operation and provides substantial improvements under congestion. In the persistent bottleneck scenario, throughput increased from 2.102 Gbit/s with fixed path forwarding to 30.0 Gbit/s with QTC, while ICMP packet loss decreased from 5% to 0%. These results indicate that QTC provides a lightweight rule based solution for responsive congestion aware traffic engineering in SDN.

I. INTRODUCTION

The internet networks provide diverse resources and software platforms for a rapidly growing number of users. Its expansion across communication, education, industry, healthcare, e-commerce and even space applications boosts the demand for continuous monitoring and efficient management [1], [2], [3]. SDN overcomes this limitation and enhances system performance by separating the control plane from the data forwarding plane enabling software based network data management through a logically centralized controller [4], [5], [6]. A key application for SDN is data flow management where traffic flow is based on service requirements and network conditions. Traffic engineering in SDN is closely linked to route discovery, traffic management, load balancing and network reliability [7], [8], [9].

Recent studies have focused on improving SDN routing and traffic distribution using optimization methods, multipath routing and QoS aware forwarding [10]. Another crucial direction is congestion aware QoS control in SDN. In which the controller node monitors the packets queue delay and redirect the path to a lower delay route when the threshold is exceeded which helps to maintain QoS under congestion conditions [11].

With the advancement of SDN, several OpenFlow controllers have been developed for both research and practical use. Early controllers such as NOX [12] presented centralized control. Researchers offer other controllers such as Beacon [13], Floodlight [14], ONOS [15], Open Daylight [16], each presents different features in terms of programmability, performance and scalability.

Among these the Ryu controller has gained attention due to its simplicity and flexibility. Ryu is an open source SDN controller developed by Nippon Telegraph and Telephone which is implemented in Python [17]. It provides well defined APIs that enable the development and network control applications. Ryu supports multiple protocols including OpenFlow, enabling its use in various SDN environments [18]. It consists of three layers: application, control and infrastructure and due to its modular design and easy to use, Ryu is widely adopted for implementing and testing SDN-based traffic management solutions.

Despite these advancements, current SDN traffic engineering and congestion management techniques

frequently rely on multiple QoS metrics, network wide state collection, delay or bandwidth estimation. These methods can be useful for route optimization, but before a forwarding decision is made, they might need more processing and wider monitoring. For situations where the goal is to identify persistent degradation on the currently active path and reroute traffic to a predetermined alternative using direct queue level congestion evidence rather than to compute a globally optimal route, a more straightforward mechanism is still required.

In order to address this gap, this study proposes a Qdisc Triggered Controller (QTC) that uses interval changes in qdisc packet drop and overlimit counters as direct evidence of congestion while only keeping an eye on the S1 uplink that is currently in use. Unlike queue monitoring methods that track instantaneous queue occupancy or depth, QTC evaluates discrete interval changes in packet drop and overlimit counters between polling instants, providing a lightweight congestion signal without continuous queue-length sampling. QTC uses queueing discipline events from the active egress interface and requires persistent congestion evidence before changing the forwarding path, unlike approaches that are based on rerouting primarily on OpenFlow traffic counters, link delay measurements, or multi metric path calculations. The suggested method combines hold down control, consecutive congestion detection, active path only monitoring, and deterministic switching between two predetermined candidate routes. In this study, lightweight refers to the use of simple rule based decision logic and limited congestion information, without network wide optimization, artificial intelligence, deep learning, or runtime multi metric route computation.

Accordingly, the aim of this study is to develop and experimentally evaluate QTC as a congestion aware SDN traffic steering mechanism capable of preserving packet delivery and service quality when the active path becomes congested. The controller is evaluated under no shaping, persistent bottleneck, transient bottleneck, and sequential bottleneck conditions to examine normal operation, congestion response, stability and switch back behavior.

II. Related Work

Several studies have addressed traffic engineering, QoS aware routing and congestion mitigation from multiple perspectives. SDN traffic engineering mechanisms were classified into topology discovery, traffic measurement, load balance and QoS in a taxonomy study [7]. Also, the study highlights challenges in scalability and mechanism design and it focuses on path selection and routing under load.

Multipath and bandwidth aware routing represent one line of related work. In [19], a multipath load balancing method that selects paths based on available bandwidth to improve delay, jitter and packet loss was proposed. The route switching mechanism relies on bandwidth based metrics which waits the event to occur then the mechanism is activated which reduces the controller efficiency. A multi metric QoS aware routing is proposed in [20], [21]. They used combined metrics such as delay, bandwidth and packet loss to optimize the routing decisions. These suggested techniques improve QoS but address route optimization rather than active path switching.

Congestion aware routing has also been investigated. The Authors in [11] suggested to trigger rerouting when queuing delay exceeds a threshold. While [22] uses link delay measurements for congestion avoidance. Both methods confirm the need for adaptive routing rather than relying on delay based or condition driven decisions.

A dynamic optimization was utilized to enhance network performance in [23], [24]. They adopt routing strategies based on network wide information such as current loads and improving jitter. Although these methods can offer flexible traffic engineering choices, they typically necessitate extra state collection and processing in order to identify the best forwarding configuration. QTC, on the other hand, only detects persistent congestion on the active uplink and switches deterministically to a predefined alternate candidate path; it does not perform runtime path calculation or network wide optimization.

In [25] forwarding behavior is chosen based on the needs of the application and the state of the network using traffic classification and centralized control. This adds extra processing for path selection and traffic identification, but it can enhance traffic allocation beyond traditional shortest path forwarding. QTC uses direct qdisc level congestion evidence to determine when the currently active path should be replaced by the predefined alternative, rather than classifying applications or calculating a traffic specific optimal route.

The reviewed studies show that SDN traffic engineering decisions can be based on available bandwidth, queueing or link delay, multiple QoS metrics, application classification, or network wide optimization. In contrast, the gap addressed in this work is stable congestion triggered switching based on direct qdisc level evidence from only the currently active uplink, without runtime route optimization, application classification, or continuous multi metric path comparison. The novelty of QTC therefore lies not merely in observing queue related information, but in combining interval changes in qdisc packet drop and overlimit counters with active path only monitoring, two consecutive congestion detections, a hold down mechanism, and deterministic switching between predefined candidate paths. The evaluation further examines no shaping, persistent, transient, and sequential bottleneck conditions to assess both congestion response and switch back behavior. Table I summarizes the main characteristics of the related approaches and highlights their differences from QTC.

TABLE I
RELATED WORK SUMMARY

Study	Main idea	Decision basis	State or processing requirement	Main distinction from QTC
[11]	Policy-based QoS rerouting	Queuing delay threshold	Queue-delay monitoring	Delay-triggered rather than qdisc event-delta triggered
[19]	Multipath load balancing	Available bandwidth	Bandwidth estimation and path comparison	Bandwidth-based rather than active-path qdisc-triggered
[20]	QoS routing	Bandwidth, delay, loss	Multiple QoS metrics	Multi-metric route selection
[21]	QoS + energy routing	Multi-constraint	Multi-constraint path evaluation	Optimization-oriented rather than congestion-triggered switching
[22]	Congestion rerouting	Link delay (LLDP)	Delay monitoring	Delay-based rather than qdisc event-based
[23]	Routing optimization	Global optimization	Network-wide optimization	Requires broader state and optimization
[24]	TE evaluation	Path + traffic split	System-level optimization	Focuses on traffic allocation rather than active-path switching
[25]	Application-aware TE	Utilization, delay	Traffic classification and centralized path selection	Requires application awareness
Present Work	Congestion-triggered candidate	qdisc drop and overlimit deltas	Active-uplink monitoring and simple rule-based decision logic	Active path qdisc triggering with deterministic alternate-path switching

III. PROPOSED METHOD

This study proposes a lightweight Qdisc Triggered Controller (QTC) for congestion aware traffic steering in a Software Defined Networking (SDN) environment. The method is intended for a topology where two predetermined candidate paths allow communication between a source and a destination. Rather than continuously collecting network-wide state information or performing runtime route optimization, QTC monitors queueing discipline statistics only on the currently active uplink and switches traffic to the alternate candidate path when persistent congestion is detected.

As shown in Figure 1, the experimental topology consists of two end hosts, H1 and H2, and four OpenFlow switches, S1, S2, S3, and S4. H2 is connected to S4, and host H1 is connected to S1. Between the hosts, two bidirectional candidate paths are predetermined. H1→S1→S2→S4→H2 is the first path, also known as the path via S2. On the other hand, H1→S1→S3→S4→H2 is followed by the second path, also known as the path via S3. The reverse traffic travels in the opposite direction along the same chosen route. At controller startup, the path via S2 is selected as the initial active path, while the path via S3 serves as the alternate candidate path.

The two S1 uplinks, s1-eth2 and s1-eth3, correspond to the paths via S2 and S3, respectively. QTC monitors only the interface associated with the currently active path. Therefore, when traffic is forwarded through S2, the controller monitors s1-eth2. However, after traffic is switched to S3, monitoring automatically moves to s1-eth3. This active path only monitoring strategy avoids continuous comparison of traffic rates across both active and inactive branches.

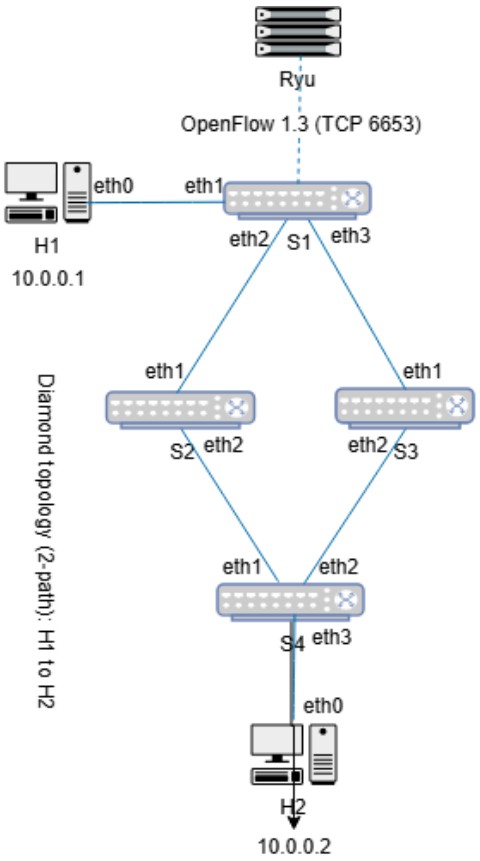

Figure 1. Experimental SDN diamond topology with two candidate paths

The paths via S2 and S3 are represented by the two S1 uplinks, s1-eth2 and s1-eth3 respectively. Only the interface connected to the active path is monitored by QTC. Thus, the controller keeps an eye on s1-eth2 when traffic is routed through S2. However, monitoring automatically switches to s1-eth3 once traffic is moved to S3. This active path only monitoring approach restricts the amount of congestion data needed for the switching decision and prevents ongoing comparisons of traffic rates between active and inactive branches.

The controller polls the qdisc statistics of the active interface every second (1s) and reads two cumulative counters, packet drops and overlimit events. Because these counters accumulate over time, QTC evaluates their changes between consecutive polling instants rather than their absolute values. Let D_t and O_t represent the cumulative numbers of packet drops and overlimit events at polling instant (t), respectively. Their interval changes are calculated as:

$$\Delta \mathrm{Dt} = \max(0, \mathrm{Dt} - \mathrm{Dt} - 1) \quad (1)$$

$$\Delta \mathrm{Ot} = \max(0, \mathrm{Ot} - \mathrm{Ot} - 1) \quad (2)$$

A polling interval is classified as congested when either the increase in overlimit events reaches 50 or at least one new packet drop occurs.

$$\mathrm{Ct} = \begin{cases} 1, & if\ (\Delta Ot \geq 50) \vee (\Delta Dt \geq 1) \\ 0, & otherwise \end{cases} \quad (3)$$

Where C_t represents congestion, A single congested observation does not trigger rerouting. QTC requires Ct = 1 for two consecutive polling intervals. If no congested observation occurs before this requirement is satisfied, the congestion streak is reset. Once two consecutive congested polls are detected, the controller verifies that the 3s hold-down interval has expired before initiating another path change. The consecutive detection requirement reduces sensitivity to isolated qdisc events, while the hold down mechanism limits rapid path oscillation after rerouting. Because two candidate paths are predefined, QTC does not calculate a new route after congestion is detected. If the path via S2 is congested, the controller will switch to the path via S3 and vice versa.

When switching is triggered, the controller updates its active path state, resets the congestion streak, and reinstalls the experiment specific OpenFlow rules across the connected switches. The table miss rule is left unchanged, but all bidirectional IPv4 forwarding rules between H1 and H2 are removed. The predefined forwarding entries linked to the newly chosen candidate path are then installed by the controller, and these rules stay in effect until a later controller initiated path change.

Accordingly, only the selected candidate path contains the experiment specific forwarding rules required for H1-to-H2 and H2-to-H1 communication. A switch from S2 path to S3 path therefore redirects traffic through S1, S3, and S4. Under sequential bottleneck conditions, if the path via S3 later becomes congested after expiry of the hold-down interval, the same mechanism permits traffic to return to S2 path.

As summarized in Figure 2, QTC determines the active candidate path, reads qdisc statistics from the corresponding S1 uplink, computes interval changes in packet drops and overlimit events, assesses the congestion condition, needs two consecutive positive detections, and, once the hold-down interval has expired, switches to the alternate candidate path and reinstalls the corresponding OpenFlow forwarding rules.

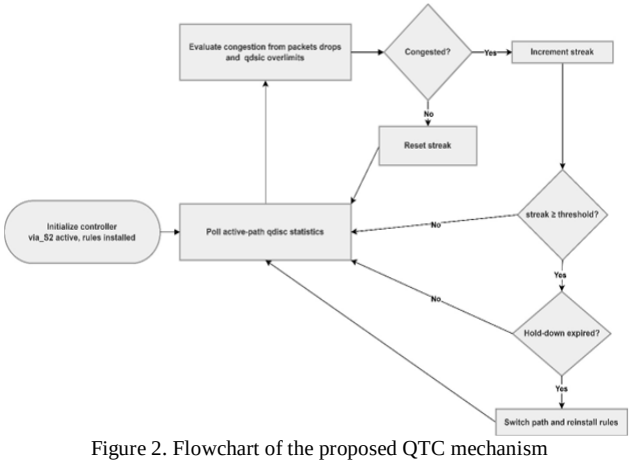

Figure 2. Flowchart of the proposed QTC mechanism

IV. EXPERIMENTAL SETUP

The QTC controller and the fixed path baseline were evaluated in an emulated SDN environment using Ubuntu 22.04.5 LTS with Mininet (version 2.3.0). In addition, the system has Open vSwitch 2.17.9 and Ryu 4.34 with OpenFlow 1.3 support. The controller applications were developed in Python 3.10. The TCP throughput and end to end delay were measured using iperf2 and ping, respectively. Controlled bottlenecks were introduced to the system using Linux tc (traffic control command) and TBF (Token Bucket Filter) queue shaping.

Four OpenFlow switches and two hosts connected via the two candidate paths outlined in Section III made up the experimental topology. To create a sustained traffic load, four parallel iperf2 streams were used to generate TCP traffic from H1 to H2. In order to examine two consecutive path degradation events, the sequential bottleneck experiment lasted 180 seconds, whereas the standard experimental runs lasted 120 seconds. Throughout each run, periodic ICMP echo measurements were taken to track packet loss and round trip time during normal operation, congestion, and recovery.

TBF shaping was applied to the active uplink of S1 with a rate of 2 Mbit/s, a burst size of 4 kB, and a latency of 100 ms to introduce controlled congestion. In the persistent bottleneck scenario, shaping was applied to s1-eth2, which corresponds to the path through S2, and remained active until the end of the run. In order to simulate transient network degradation, shaping was added to the transient bottleneck scenario following a brief period of regular operation before being removed. The S2 path was first affected in the sequential bottleneck scenario. After that, shaping was applied to s1-eth3 instead of S2, enabling an assessment of QTC's return switching behavior.

Every QTC experiment used the same configuration for switching and monitoring. Every second, the controller reads the active S1 uplink's qdisc statistics. When there was at least one new packet drop or 50 newly recorded overlimit events, a polling interval was deemed congested. Before the controller could start a path change, there had to be two consecutive congested polling intervals. A three second (3s) hold down period followed each switch to stop another instantaneous path change. The monitored interface automatically switched between s1-eth2 and s1-eth3 in accordance with the active route, which was initially the path through S2.

During each run, multiple measurement sources were captured to analyze controller behavior and network performance. These included OpenFlow port counters from S1, QTC controller logs, qdisc statistics from the shaped interfaces, periodic ping measurements, and iperf2 client output. While the OpenFlow port counters were utilized to verify the distribution of traffic between the two candidate paths, the qdisc statistics offered details about packet drops and overlimit events. The monitored interface, variations in qdisc counters between polls, congestion detection streaks, and path switch events were all noted in the controller logs. The observed reaction between the start of congestion and the associated path change could be identified thanks to these logs.

Four scenario classes were used to structure the evaluation: No Shaping (NS), Persistent Bottleneck (PB), Transient Bottleneck (TB), and Sequential Bottleneck (SB). Three scenarios were used to evaluate the fixed path baseline: no shaping, persistent bottleneck, and transient bottleneck. Additionally, QTC was assessed in the sequential bottleneck scenario to confirm switching between the two candidate routes in both directions. To ensure consistency in the comparison across scenarios, the same controller configuration was kept constant throughout all QTC runs. A summary of all the experiments is provided in Table II.

TABLE II
STUDY EXPERIMENT SCENARIOS

Run	Controller	Scenario	Purpose
1	Baseline	No Shaping	Normal-operation reference
2	Baseline	Persistent S2 bottleneck	Fixed-path stress case
2b	Baseline	Transient S2 bottleneck	Fixed-path realistic congestion case
3	Proposed	No Shaping	Adaptive-controller normal-operation case
4	Proposed	Persistent S2 bottleneck	Adaptive-controller stress case
4b	Proposed	Transient S2 bottleneck	Adaptive-controller realistic congestion case
5	Proposed	Sequential bottlenecks	Switch-back validation

These runs were designed to evaluate normal operation, sustained degradation, temporary congestion and sequential bottleneck conditions under both fixed-path and adaptive-path control.

V. RESULTS AND DISCUSSION

Results were obtained for four scenarios: No Shaping (NS), Persistent Bottleneck (PB), Transient Bottleneck (TB), and Sequential Bottleneck (SB) for both the proposed QTC controller and the fixed path baseline (BL). Overall throughput, average RTT, RTT mean deviation, end to end ICMP packet loss, and queue behavior obtained from qdisc statistics were used to assess performance. Additionally, congestion detection and path switching behavior were examined using OpenFlow port statistics and controller logs.

The performance proposed adaptive mechanism was compared to the No Shaping scenario to verify that it does not degrade normal network operation. Accordingly, the fixed path BL achieved 32.2 Gbit/s, while the QTC controller achieved 32.3 Gbit/s as presented in Figure 3. In both cases the ICMP packets loss remained low at 0% and the average RTT stayed below 0.05 ms. In addition, qdisc reported zero packets drop with no overlimit events detected, indicating the absence of significant queue buildup. At S2 the OpenFlow port statistics further confirmed that the traffic remained entirely via the S2 path in both scenarios. While the controller logs showed no switch event.

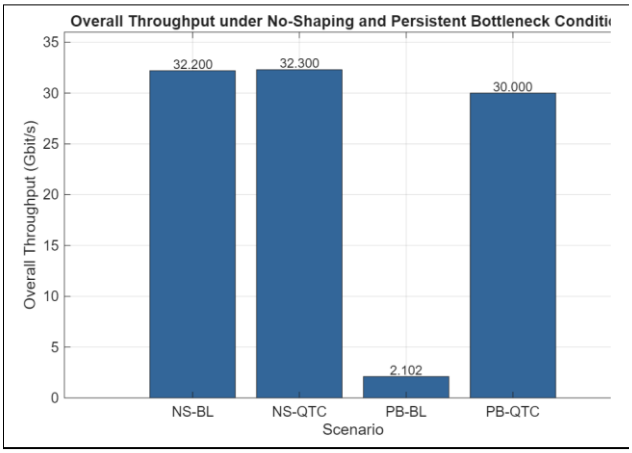

Figure 3 Throughput comparison under No Shaping (NS) and persistent bottleneck (PB) conditions for fixed path baseline (BL) and the proposed controller (QTC).

Under the persistent S2 bottleneck case a distinct difference emerged. With the S2 path continuously shaped the BL remained on the degraded path with 2.102 Gbit/s throughput. While the QTC controller rerouted the traffic through the alternate S3 path achieving 30 Gbit/s as illustrated

in Figure 3. The BL also recorded 5% ICMP packets loss and the RTT increased to approximately 109.6 ms and 106.2 ms in the final sampled intervals. In comparison, the QTC controller maintained 0% packet loss with 0.04 to 0.05 ms of RTT values. These findings show that QTC preserves near baseline performance during normal operation and substantially improves throughput, delay, and packet delivery when a persistent bottleneck affects the active path.

The queue level results investigated in Figure 4 which indicates the network performance differences. The fixed path BL accumulated 37,751 dropped packets and 131,948 overlimit events on the shaped S2 uplink. Whereas the QTC controller reduced these values to 21,942 and 5,146 respectively which presents significant decrease in queued packets due to path adaptation.

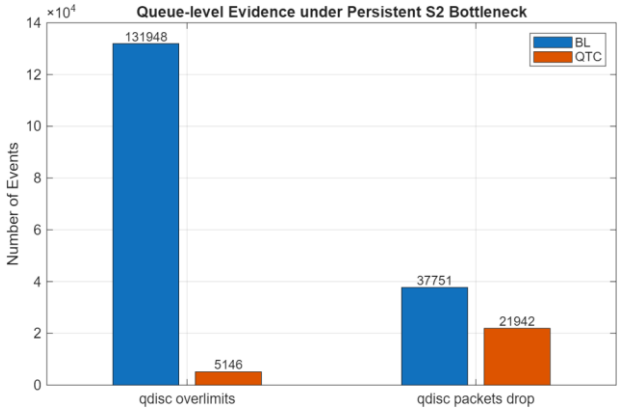

Figure 4 Queue level packets drop and overlimit events under persistent S2 bottleneck.

The OpenFlow port findings at S1 validate the corresponding forwarding behavior. First, the BL case the traffic remained concentrated on the S2 path with port 2 transmitting 31.52 GB while port 3 managed to send 13.39 MB only. In contrast, the QTC controller shifted traffic to the alternate path which led to port 3 transmitting 355.95 GB compared to 25.09 GB on port 2. Controller logs showed that shaping was applied at 14:22:57, followed by two consecutive congested polls, after which QTC switched from the S2 path to the S3 path at 14:22:59. The observed congestion detection and path switching response was therefore approximately 2 s.

The transient bottleneck scenario was utilized to emulate temporary congestion event on the active S2 route. During the shaping period the fixed path technique stayed on the degraded path whereas the QTC controller detects this degradation and routes packets delivery through S3. Under this scenario, the BL achieved about 16.7 Gbit/s while the QTC controller delivered 30.7 Gbit/s as shown in Figure 5.

A direct comparison during the actual congestion interval provides a clearer view of the improvement. During the 15 to 60 s shaping period, the fixed path baseline achieved an average throughput of 5.810 Gbit/s, whereas QTC maintained 30.656 Gbit/s. This corresponds to approximately 5.3 times the throughput of the baseline during the congestion period.

Figure 6 shows the queue level observation, during the shaping interval on the S2 uplink the BL accumulated 28,016

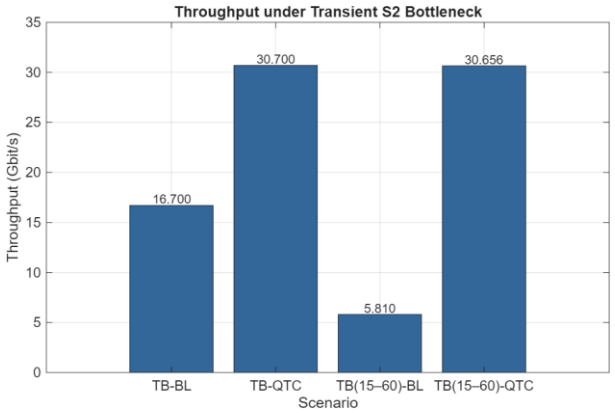

Figure 5. Throughput comparison under transient bottleneck (TB) conditions for fixed path baseline (BL) and the proposed controller (QTC).

dropped packets and recorded 89,218 overlimit events. While the QTC reduced those values to 20,265 and 3,401 respectively. This indicates lower queue stacking under adaptive path selection.

The OpenFlow statistics at port S1 further confirms these findings. In the BL scenario the traffic remained on the S2 path with port 2 transmitting 232.67 GB and port 3 only send 23.32 MB. In contrast, the QTC shifted the traffic to the second path through port 3 which led to transmitting 351.8 GB compared to 87.26 GB on port 2. Controller logs show that after shaping was applied the QTC controller detected queue accumulation and switched the route from via S2 to via S3 within two seconds.

Figure 7 shows the sequential bottleneck experiment, where the proposed QTC controller initially operated on the

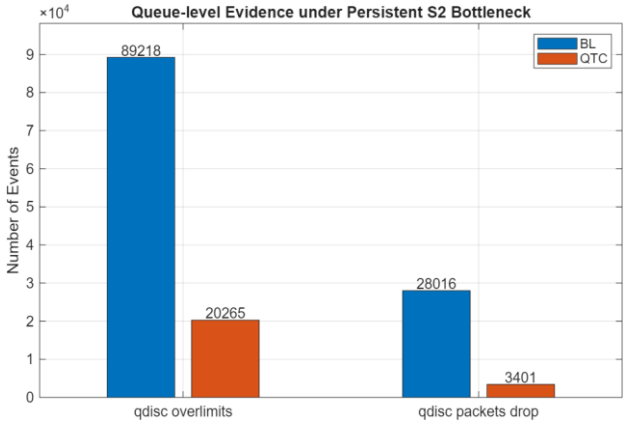

Figure 6. Queue level packet drops and overlimit events under transient S2 bottleneck

S2 path, switched to S3 after the S2 route performance degradation. Later returned to transmit packets through S2 when S3 became congested. In the sequential bottleneck experiment, QTC initially operated on the S2 path and switched to S3 after congestion was detected on S2.

Later, when shaping was removed from S2 and applied to S3, the controller detected congestion on the new active path and returned traffic to S2. The first switch occurred approximately 1s after the S2 shaping event, while the return switch occurred approximately 2s after shaping was applied to S3. In both cases, switching occurred only after the required consecutive congestion detections, and no repeated path oscillation was observed during the experiment.

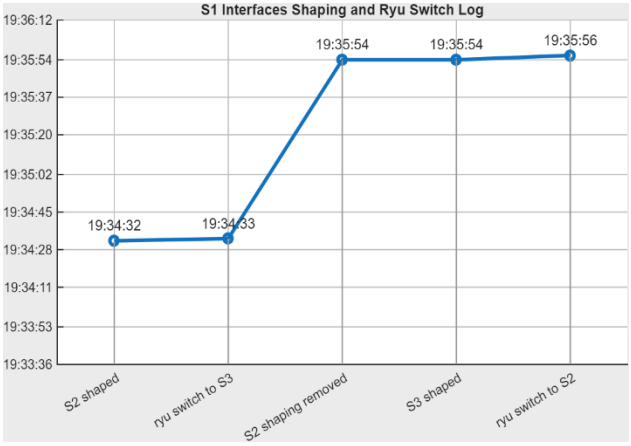

Figure 7. Path switching timeline under sequential S2/S3 bottleneck.

The QTC technique achieves near BL performance under normal conditions while keeping high throughput and low latency during persistent and transient bottlenecks. It mitigates congestion by employing path switching and eliminating packet loss and reducing queue buildup. The sequential bottleneck findings show robust adaptation and stable performance under dynamic network conditions.

Table III summarizes the throughput, packet loss, RTT, and queue level results across all evaluated scenarios for both the fixed path baseline and the QTC controller.

In the persistent bottleneck run, the QTC controller process consumed an average of 10% CPU and around 150 MB of memory on the experimental host, with peak usage occurring at the moment of path switching. Each switching event generated single-digit extra OpenFlow messages to update forwarding rules across the four switches.

TABLE III
SUMMARY OF KEY PERFORMANCE METRICS

Scenario	Controller	Throughput (Gbit/s)	Packet Loss	Avg RTT	qdisc Drops	Overlimit Events
NS	BL	32.2	0%	< 0.05 ms	0	0
NS	QTC	32.3	0%	< 0.05 ms	0	0
PB	BL	2.102	5%	~106–110 ms	37,751	131,94
PB	QTC	30.0	0%	0.04–0.05 ms	21,942	5,14
TB	BL	16.7	0%	Peak average 108.9 ms	28,016	89,21
TB	QTC	30.7	0%	Peak average 0.082 ms	20,265	3,401

During normal operation, the no-shaping results confirm that QTC does not impose a measurable performance penalty, as throughput, packet loss, and delay remained equivalent to the fixed-path baseline. The observed convergence time, defined as the interval between congestion onset and path switching completion, was approximately 2s in the persistent and transient scenarios and between 1s and 2s in the sequential bottleneck, consistent with the two consecutive 1s polling intervals required before switching is triggered. Scalability with respect to the number of switches, hosts, or concurrent flows was not assessed in this study and remains a direction for future work.

The results also indicate that the consecutive detection requirement and hold down mechanism were effective in preventing immediate repeated switching during the evaluated scenarios. No unnecessary path changes were observed in the No Shaping experiment, while the sequential bottleneck test demonstrated controlled switching in both directions as the active bottleneck changed. However, packet reordering was not directly measured, and transient reordering during active flow redirection cannot therefore be excluded. In addition, the present evaluation was limited to the considered Mininet topology and fixed controller configuration, so broader scalability, sensitivity, and physical testbed validation remain subjects for future investigation.

VI. CONCLUSION

This paper introduced a lightweight Qdisc Triggered Controller (QTC) for congestion aware traffic steering in SDN. By using qdisc level indicators together with consecutive congestion detections and a hold down mechanism, the proposed controller responds to persistent congestion while limiting frequent path changes. The implementation and evaluation in a Mininet and Ryu environment showed that QTC maintained stable forwarding behavior and improved throughput, latency, and packet delivery compared with fixed path forwarding under the evaluated bottleneck conditions. The findings demonstrate that effective congestion aware traffic steering can be achieved using simple rule based decision logic without network wide optimization or runtime multi metric route computation. Future work will investigate larger topologies, additional traffic conditions, sensitivity to controller parameters, resource overhead, and validation in a physical network testbed.

REFERENCES

[1] J. P. Simon, "The global internet market(s): a reconstruction of the views of the industry," *Digit. Policy Regul. Gov.*, vol. 22, no. 2, pp. 109–133, May 2020, doi: 10.1108/DPRG-10-2019-0092.

[2] Z. S. K. Al-Shammari, "Power Minimisation Techniques for Space-Based Wireless Sensor Networks," thesis, University of Leicester, 2017. Accessed: May 01, 2026. [Online]. Available: https://figshare.le.ac.uk/articles/thesis/Power_Minimisation_Techniques_for_Space-Based_Wireless_Sensor_Networks/10199786/1

[3] A. A. Al-Shammaa and A. J. Stocker, "Distributed clusters classification algorithm for indoor wireless sensor networks using pre-defined knowledge-based database," in *2019 IEEE Sensors Applications Symposium (SAS)*, Sophia Antipolis, France: IEEE, Mar. 2019, pp. 1–6. doi: 10.1109/SAS.2019.8706099.

[4] K. Nisar *et al.*, "A survey on the architecture, application, and security of software defined networking: Challenges and open issues," *Internet Things*, vol. 12, p. 100289, Dec. 2020, doi: 10.1016/j.iot.2020.100289.

[5] E. R. Jimson, K. Nisar, and Mohd. H. Bin Ahmad Hijazi, "Bandwidth management using software defined network and comparison of the throughput performance with traditional network," in *2017 International Conference on Computer and Drone Applications (IConDA)*, Kuching: IEEE, Nov. 2017, pp. 71–76. doi: 10.1109/ICONDA.2017.8270402.

[6] H. Babbar, S. Rani, and S. A. AlQahtani, "Intelligent Edge Load Migration in SDN-IIoT for Smart Healthcare," *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 8058–8064, Nov. 2022, doi: 10.1109/TII.2022.3172489.

[7] R. Mohammadi, S. Akleylek, A. Ghaffari, and A. Shirmarz, "Taxonomy of traffic engineering mechanisms in software-defined networks: a survey," *Telecommun. Syst.*, vol. 81, no. 3, pp. 475–502, Nov. 2022, doi: 10.1007/s11235-022-00947-6.

[8] S. J. Rajan *et al.*, "Efficient traffic management with adaptive SDN in vehicular networks," *Sci. Rep.*, vol. 15, no. 1, p. 11785, Apr. 2025, doi: 10.1038/s41598-025-96365-0.

[9] A. B. Vieira, W. N. Paraizo, L. J. Chaves, L. H. A. Correia, and E. F. Silva, "An SDN-based energy-aware traffic management mechanism," *Ann. Telecommun.*, vol. 77, no. 3–4, pp. 139–150, Apr. 2022, doi: 10.1007/s12243-021-00863-x.

[10] R. Farahi, "A comprehensive overview of load balancing methods in software-defined networks," *Discov. Internet Things*, vol. 5, no. 1, p. 6, Jan. 2025, doi: 10.1007/s43926-025-00098-5.

[11] I. Ali, S. Hong, and T. Cheung, "Quality of Service and Congestion Control in Software-Defined Networking Using Policy-Based Routing," *Appl. Sci.*, vol. 14, no. 19, p. 9066, Oct. 2024, doi: 10.3390/app14199066.

[12] N. O. X. Repo, *noxrepo/nox*. (Feb. 24, 2026). C++. Accessed: May 05, 2026. [Online]. Available: https://github.com/noxrepo/nox

[13] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, Hong Kong China: ACM, Aug. 2013, pp. 13–18. doi: 10.1145/2491185.2491189.

[14] *floodlight/floodlight*. (Apr. 22, 2026). Java. Project Floodlight. Accessed: May 05, 2026. [Online]. Available: https://github.com/floodlight/floodlight

[15] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, Chicago Illinois USA: ACM, Aug. 2014, pp. 1–6. doi: 10.1145/2620728.2620744.

[16] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, Sydney, Australia: IEEE, Jun. 2014, pp. 1–6. doi: 10.1109/WoWMoM.2014.6918985.

[17] *faucetsdn/ryu*. (May 01, 2026). Python. Faucet SDN. Accessed: May 05, 2026. [Online]. Available: https://github.com/faucetsdn/ryu

[18] Md. T. Islam, N. Islam, and Md. A. Refat, "Node to Node Performance Evaluation through RYU SDN Controller," *Wirel. Pers. Commun.*, vol. 112, no. 1, pp. 555–570, May 2020, doi: 10.1007/s11277-020-07060-4.

[19] F. Chahlaoui, H. Dahmouni, and H. El Alami, "Multipath-routing based load-balancing in SDN networks," in *2022 5th Conference on Cloud and Internet of Things (CIoT)*, Marrakech, Morocco: IEEE, Mar. 2022, pp. 180–185. doi: 10.1109/CIoT53061.2022.9766801.

[20] I. E. Kamarudin and M. A. Ameedeen, "QSroute: Enhancing Software Defined Networking Routing Scheme through Advanced QoS Metrics Integration," *JOIV Int. J. Inform. Vis.*, vol. 9, no. 4, p. 1697, Jul. 2025, doi: 10.62527/joiv.9.4.3025.

[21] Y. Zhang and S. Gorlatch, "Optimizing Energy Efficiency of QoS-Based Routing in Software-Defined Networks," in *Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Alicante Spain: ACM, Nov. 2021, pp. 87–94. doi: 10.1145/3479242.3487325.

[22] J.-H. Sung and Y.-C. Lin, "Software-Defined Network Congestion Mitigation Strategy Based on Link," in *Proceedings of the 2024 12th International Conference on Computer and Communications Management*, Kagoshima-shi Japan: ACM, Jul. 2024, pp. 86–92. doi: 10.1145/3688268.3688282.

[23] J. Chen, W. Xiao, H. Zhang, J. Zuo, and X. Li, "Dynamic routing optimization in software-defined networking based on a metaheuristic algorithm," *J. Cloud Comput.*, vol. 13, no. 1, p. 41, Feb. 2024, doi: 10.1186/s13677-024-00603-1.

[24] M. I. Salman and B. Wang, "Boosting performance for software defined networks from traffic engineering perspective," *Comput. Commun.*, vol. 167, pp. 55–62, Feb. 2021, doi: 10.1016/j.comcom.2020.12.018.

[25] S. Jeong, D. Lee, J. Hyun, J. Li, and J. W.-K. Hong, "Application-aware traffic engineering in software-defined network," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2017, pp. 315–318. doi: 10.1109/APNOMS.2017.8094144.