On How to Provision Virtual Circuits for Network-Redirected Large-Sized, High-Rate Flows

Zhenzhen Yan, Malathi Veeraraghavan
Dept. of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA, USA
Email: {zy4d,mvee}@virginia.edu

Chris Tracy, Chin Guok
Energy Sciences Network (ESnet)
LBNL
Berkeley, CA, USA
Email: {ctracy,chin}@es.net

***Abstract*—To reduce the impact of large-sized, high-rate (α) transfers on real-time flows, a Hybrid Network Traffic Engineering System (HNTES) was proposed in earlier work. HNTES is an intra-domain solution that enables the automatic identification of α flows at a provider network's ingress routers, and redirects these flows to traffic-engineered QoS-controlled virtual circuits. The purpose of this work is to determine the best QoS mechanisms for the virtual circuits used in this application. Our findings are that a no-policing, two-queues solution with weighted fair queueing and priority queueing is both sufficient and the best for this application. It allows for the dual goals of reduced delay/jitter in real-time flows, and high-throughput for the α flows, to be met.**

Keywords*—policing; scheduling; high-speed networks; traffic-engineering; virtual-circuit networks

I. Introduction

For large-sized scientific dataset transfers, scientists typically invest in high-end computing systems that can source and sink data to/from their disk systems at high speeds. These transfers are referred to as α flows as they dominate other flows [1]. They also cause increased burstiness, which in turn impacts delay-sensitive real-time audio/video flows. In prior work [2], we proposed an overall architecture for an intra-domain traffic engineering system called Hybrid Network Traffic Engineering System (HNTES) that performs two tasks: (i) analyzes NetFlow reports offline to identify α flows, and (ii) configures the ingress routers for future α-flow redirection to traffic-engineered Quality-of-Service (QoS)-controlled paths. The prior paper [2] then focused on the first aspect, and analyzed NetFlow data obtained from live ESnet routers for the period May to Nov. 2011. The analysis showed that since α flows require high-end computing systems to source/sink data at high speeds, these systems are typically assigned static global public IP addresses, and repeated α flows are observed between the same pairs of hosts. Therefore source and destination address prefixes of observed α flows can be used to configure firewall filter rules at ingress routers for future α-flow redirection. The effectiveness of such an offline α-flow identification scheme was evaluated with the collected NetFlow data and found to be 94%, i.e., a majority of bytes sent in bursts by α flows would have been successfully isolated had such a traffic engineering system been deployed [2].

The work presented here focuses on the second aspect of HNTES by addressing the question of how to achieve α-flow redirection and isolation to traffic-engineered paths. Specifically, service providers such as ESnet [3] are interested in actively selecting traffic-engineered paths for α-flows, and using QoS mechanisms to isolate these flows. With virtual-circuit technologies, such as MultiProtocol Label Switching (MPLS), ESnet and other research and education network providers, such as Internet2, GEANT [4], and JGN-X [5], offer a dynamic circuit service. An On-Demand Secure Circuits and Advance Reservation System (OSCARS) Inter-Domain Controller (IDC) [6] is used for circuit scheduling and provisioning.

The basic interface to the IDC requires an application to specify the circuit rate, duration, start time, and the endpoints in its advance-reservation request. The specified rate is used both for (i) path computation in the call-admission/circuit-scheduling phase and (ii) policing traffic in the data plane. If the application requests a high rate for the circuit, the request could be rejected by the OSCARS IDC due to a lack of resources. On the other hand, if the request is for a relatively low rate (such as 1 Gbps), then the policing mechanism could become a limiting factor to the throughput of α flows, preventing TCP from increasing its sending rate.

The purpose of this paper is to evaluate the effects of different scheduling and policing mechanisms to achieve two goals: (i) reduce delay and jitter of real-time sensitive flows that share the same interfaces as α flows, and (ii) achieve high throughput for α-flow transfers.

Our *key findings* are as follows: (i) With the current widely deployed best-effort IP-routed service, which uses first-come-first-serve (FCFS) packet scheduling on egress interfaces of routers, the presence of an α flow can increase the delay and jitter experienced by audio/video flows. (ii) This influence can be eliminated by configuring two virtual queues at the contending interface and redirecting identified α flows to one queue (α queue), while all other flows are directed to a second queue (β queue). (iii) The policer should not be configured to direct out-of-profile packets of an α TCP flow to a different queue from its in-profile packets. When packets of the same TCP flow are served from different queues, packets can arrive out of sequence at the receiver. Out-of-sequence arrivals triggers TCP's fast retransmit/fast recovery congestion algorithm, which causes the TCP sender to lower its sending rate resulting in degraded throughput. (iv) An alternative approach to dealing with out-of-profile packets is to probabilistically drop a few packets using Weighted Random Early Detection (WRED), and to buffer the remaining out-of-profile packets in the same queue as the in-profile packets. This prevents the out-of-sequence problem and results in a smaller drop in α-flow throughput when compared to the separate-queues approach. (v) The no-policing scheme is preferred to the policing/WRED scheme because HNTES redirects α flows within a provider's network, which means that these flows will typically run TCP and are not rate-limited to the circuit rate. If an end application requested a circuit explicitly, then it can be expected to use traffic control mechanisms, such as Linux `tc`, to limit the sending rate. But with HNTES, the end application is not involved in the circuit setup phase, and therefore the applications are likely to be running unfettered TCP. Under these conditions, when buffer occupancy builds up, packets will be deliberately dropped in the policing/WRED scheme, leading to poor performance. Furthermore, if there are two simultaneous α flows, the probability of buffer buildups increases, which in turn increases the dropped-packet rate and lowers throughput. (vi) The negatives of partitioning rate/buffer space resources between two queues were studied. Our conclusions are that close network monitoring is required to dynamically adjust the rate/buffer space split between the two queues as traffic changes, and the probability of unidentified α flows should be reduced whenever possible to avoid these flows from becoming directed to the β queue.

Section II provides background and reviews related work. Section III describes the experiments we conducted on a high-speed testbed to evaluate different combinations of QoS mechanisms and parameter values to achieve our dual goals of reduced delay/jitter for real-time flows and high throughput for α flows. Our conclusions are presented in Section IV.

II. Background and related work

The first three topics, historical perspective, a hybrid network traffic engineering system, and QoS support in state-of-the-art routers, provide the reader with relevant background information. The last topic, QoS mechanisms applied to TCP flows, covers related work.

Historical perspective: In the nineties, when Asynchronous Transfer Mode (ATM) [7] and Integrated Services (IntServ) [8] technologies were developed, virtual circuit (VC) services were considered for delay-sensitive multimedia flows. However, these solutions are not scalable to large numbers of flows because of the challenges in implementing QoS mechanisms such as policing and scheduling on a per-flow basis. Instead, a solution of overprovisioning connectionless IP networks has been affordable so far. Overprovisioning prevents router-buffer buildups and thus ensures low delay/jitter for real-time audio/video flows. While this solution works well most of the time, there are occasional periods when a single large dataset transfer is able to ramp up to a very high rate and adversely affect other traffic [9]. Such transfers, which are referred to as α flows, occur when the amount of data being moved is large, and the end-to-end sustained rate is high.

In the last ten years, there has been an emergent interest in using VCs but for α-flow transfers not multimedia flows. As noted in Section I, service providers are interested in routing these α flows to traffic-engineered, QoS-controlled paths. The scalability issue is less of a problem here since the number of α flows is much smaller than of that of real-time audio-video flows. It is interesting to observe this "flip" in the type of applications being considered for virtual-circuit services, i.e., from real-time multimedia flows to file-transfer flows.

Hybrid Network Traffic Engineering System (HNTES): Ideally if end-user applications such as GridFTP [10] alerted the provider networks en route between the source and destination before starting a high-rate, large-sized dataset transfer, these networks could perform path-selection and direct the resulting TCP flow(s) to traffic-engineered, QoS-controlled paths. However, most end-user applications do not have this capability, and furthermore inter-domain signaling to establish such paths requires significant standardization efforts. Meanwhile, providers have recognized that intra-domain traffic-engineering is sufficient if α flows can be automatically identified at the ingress routers. Deployment of such a traffic-engineering system lies within the control of individual provider networks, making it a more attractive solution. Therefore, the first step in our work was to determine whether such automatic α flow identification is feasible or not.

In our prior work [2], we started with a hypothesis that computers capable of sourcing/sinking data at high rates are typically allocated static public IP addresses, and α flows between pairs of these computers occur repeatedly as the same users initiate dataset transfers. This hypothesis was true for ESnet traffic. Therefore HNTES can determine source-destination IP address prefixes by analyzing NetFlow reports of completed α flows and use these address prefixes to set firewall filters to redirect future α flows. Our heuristic was simple: if a NetFlow report for a flow showed that more than H bytes (set to 1 GB) were sent within a fixed time interval (set to 1 min), we classified the flow as an α flow. This NetFlow data analysis is envisioned to be carried out offline on say a nightly basis for all ingress routers to update the firewall filters. If no flows are observed for a particular source-destination address prefix within an aging interval (set to 30 days), then the firewall filter entry is removed. The effectiveness of this scheme was evaluated through an analysis of 7 months of NetFlow data obtained from an ESnet router. For this data set, 94% (82%) of bytes generated by α flows in bursts would have been identified correctly and isolated had /24 (/32) based prefix IDs been used in the firewall filters.

QoS support in state-of-the-art routers: Multiple policing, scheduling and traffic shaping mechanisms have been implemented in today's routers. We review the particular mechanisms used in ESnet, and hence in our experiments.

For scheduling, two mechanisms are used: Weighted Fair Queueing (WFQ) and Priority Queueing (PQ) [11]. With WFQ, multiple traffic classes are defined, and corresponding virtual queues are created on egress interfaces. Bandwidth can be strictly partitioned or shared among the virtual queues. WFQ is combined with PQ as explained later. On the ingress-side, policing is used to ensure that a flow does not exceed its assigned rate (set by the IDC during call admission). For example, in a single-rate two-color (token bucket) scheme, the average rate (which is the rate specified to the IDC in the circuit request) is set to equal the generation rate of tokens, and a maximum burst-size is used to limit the number of tokens in the bucket. The policer marks packets as *in-profile* or *out-of-profile*. Three different actions can be configured: (i) discard out-of-profile packets immediately, (ii) classify out-of-profile packets as belonging to a `Scavenger Service (SS)` class, and direct these packets to an `SS` virtual queue, or (iii) drop out-of-profile packets according to a WRED profile, but store remaining out-of-profile packets in the same queue as in-profile packets. For example, the drop rate for out-of-profile packets can be configured to increase linearly from 0 to 100 for corresponding levels of queue occupancy.

QoS mechanisms applied to TCP flows: Many QoS provisioning algorithms that involve some form of active queue management (AQM) have been studied [12]–[16]. Some of the simpler algorithms have been implemented in today's routers, such as RED [12] and WRED [14], while other algorithms, such as Approximate Fair Dropping (AFD) [16], have been shown to provide better fairness. An analysis of the configuration scripts used in core and edge routers of ESnet shows that these AQM related algorithms are not enabled. This is likely due to the commonly adopted policy of overprovisioning (an Internet2 memorandum [17] states a policy of operating links at 20% occupancy). Nevertheless, providers have recognized that in spite of the headroom, an occasional α flow can spike to a significant fraction of link capacity (e.g., our GridFTP log analysis showed average flow throughput of over 4 Gbps across 10-Gbps paths [9]). When the flow throughput averaged across its lifetime is 4 Gbps, there can be short intervals in which the flow rate spiked to values close to link capacity.

III. Experiments

A set of experiments were designed and executed to determine the best combination of QoS mechanisms with corresponding parameter settings in order to achieve our dual goals of reduced delay/jitter for real-time traffic and high throughput for α flows. For the first goal, we formulated a hypothesis as follows: a scheduling-only no-policing scheme that isolates α-flow packets into a separate virtual queue is sufficient to keep non-α flow delay/jitter low. For the second goal, we experimented with different QoS mechanisms and parameter settings.

Experiment 1 was designed to understand the two modes for sharing link rate (strictly partitioned and work conversing), and to determine the router buffer size. *Experiment 2* tests the above-stated hypothesis for the first goal. *Experiments 3 and 4* studied two different mechanisms, using a separate scavenger-service (SS) queue vs. using WRED, for handling the out-of-profile packets identified by ingress-side policing, and compared results with a no-policing approach. We concluded that the WRED scheme was better, but it was outperformed by the no-policing scheme. *Experiment 5* was designed to check if the policing/WRED scheme had a fairness advantage over the no-policing scheme. We found that since neither of the two policed α flows honored their assigned rates (which should be expected for HNTES-redirected flows), under the no-policing scheme the TCP flows adjusted their sending rates and had no packet losses, while the deliberate packet losses introduced in the policing/WRED scheme lowered throughput for both flows, and furthermore resulted in lower fairness because of a difference in RTTs, even though this difference was small. In *Experiment 6*, we characterized the the impact of QoS provisioning under changing traffic conditions, and compared two versions of TCP: Reno and H-TCP. In the presence of an α flow that uses up its whole α-queue rate allocation, if the background traffic is more than the rate allocated to the β queue, the latter will suffer from more losses than if there had been no partitioning of resources between the two queues. This implies a need for closer monitoring of traffic and dynamic reconfiguration of the rate/buffer allocations to the two queues. However, since two rare events, an α flow and an increased background load, have to occur simultaneously, the probability of this scenario is low. H-TCP is better than Reno for high-speed transfers, but from the perspective of the impact on other flows, we did not see a significant difference in our tested scenarios. *Experiment 7* was designed to study the effects of an unidentified α flow being directed to the β queue. Here again, if there was no simultaneous α flow directed to the α queue when the unidentified α flow appeared, then the impact will be the same as without partitioning. However, if this combination of rare events occurs jointly, then given that the β queue has only a partition of the total interface rate/buffer space, the impact on delay-sensitive flows will be greater than if there had been no partitioning.

Section III-A describes the experimental setup, the experimental methodology, and certain router configurations that are common to all the experiments. The remaining subsections describe the seven experiments.

A. Experimental setup

The experimental network setup is shown in Figure 1. It was called the Long Island MAN (LIMAN) testbed, and was supported by ESnet as a DOE-funded testbed for networking research. The high-performance hosts, `W1` (West 1), `E1` (East 1), and `E2` (East 2), were Intel Xeon Nehalem E5530 models (2.4GHz CPU, 24GB memory) and ran Linux version 2.6.33. The application hosts, `WA` (West App-host) and `EA` (East App-host), were Intel Dual 2.5GHz Xeon model and ran Linux 2.6.18. The routers, `WR` (West Router) and `ER` (East Router), were Juniper MX80's running Junos version 10.2. The link rates were 10 Gbps from the high-performance hosts to the

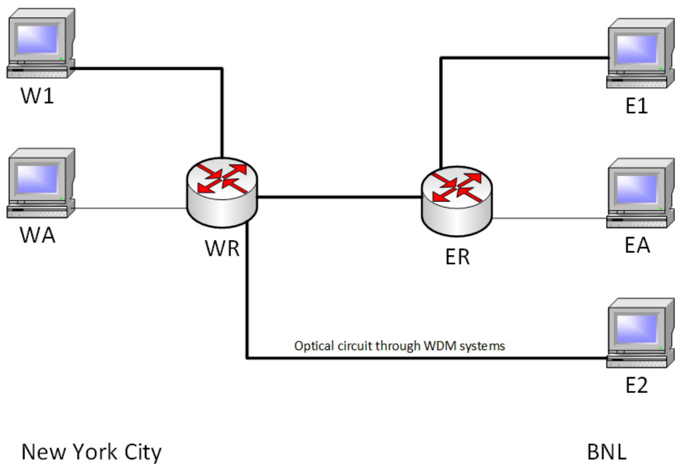


Figure 1. Experiment setup

routers, 1 Gbps from the application hosts to the routers, and 10 Gbps between the routers.

Host `W1` and router `WR` were physically located in New York City, while the East-side hosts and routers, and host `E2`, were physically located in the Brookhaven National Laboratory (BNL) in Long Island, New York. Host `E2` was connected to router `WR` via a circuit provisioned across the Infinera systems of the underlying optical network as shown in Figure 1.

Each experiment consists of four steps: (i) plan the applications required to test a particular QoS mechanism, (ii) configure routers to execute the selected QoS mechanisms with corresponding parameter settings based on the planned application flows, (iii) execute applications on end hosts to create different types of flows through the routers, and (iv) obtain measurements for various characteristics, e.g., throughput, packet loss, and delay, from the end-host applications as well as from packet counters in the routers.

A preliminary set of experiments were conducted to determine the specific manner in which the egress-side link capacity was shared among multiple virtual queues. Theoretically the transmitter can be strictly partitioned or shared in a work-conserving manner. If strictly partitioned, then even if there are no packets waiting in one virtual queue, the transmitter will not serve packets waiting in another queue. In this mode, each queue is served at the exact fractional rate assigned to it. In contrast, in the work-conserving mode the transmitter will serve additional packets from a virtual queue that is experiencing a higher arrival rate than its assigned rate if there are no packets to serve from the other virtual queues. The buffer is always strictly partitioned between the virtual queues in the routers used in our experiments.

Figure 2 illustrates how a combination of QoS mechanisms was used in our experiments. *First*, incoming packets are classified into multiple classes based on pre-configured firewall filters, e.g., α-flow packets are identified by the source-destination IP address prefixes and classified into the α class. *Second*, packets in some of these classes are directly sent to corresponding egress-side virtual queues, while flows corresponding to other classes are subject to policing. A single-rate token bucket scheme is applied. If an arriving packet finds a token in the bucket, it is marked as being in-profile; otherwise it is marked as being out-of-profile. *Third*, for some policed flows, in-profile and out-of-profile packets are sent to separate egress-side virtual queues, while packets from other policed flows are subject to WRED before being buffered in a single virtual queue. On the egress-side, each virtual queue is assigned a priority level, a fractional allocation (expressed as a percentage) of link capacity, and a fractional allocation of the buffer. As noted in the previous paragraph the buffer allocation is strictly partitioned while the transmitter is shared in work-conserving mode. *Fourth*, the WFQ scheduler decides whether a virtual "queue is in-profile or not," by comparing the rate allocated to the queue and the rate at which packets have been served out of the queue. *Finally*, the PQ scheduler selects the queue from which to serve packets using their assigned priority levels, but to avoid starvation of low-priority queues, as soon as a large enough number of packets are served from a high-priority queue to cause the status of the queue to transition to out-of-profile, the PQ scheduler switches to the next queue in the priority ordering. When all queues become out-of-profile,

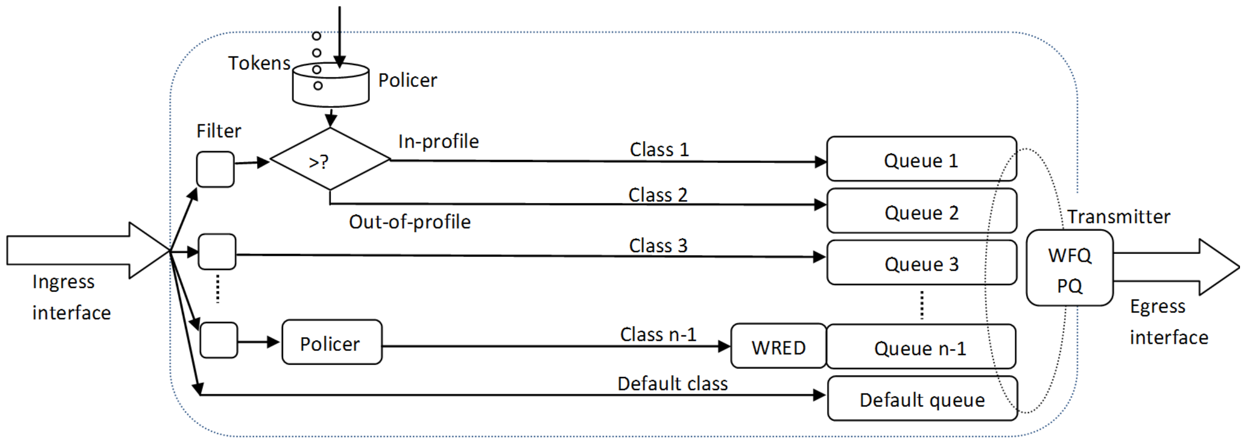


Figure 2. Illustration of QoS mechanisms in a router

it starts serving packets again in priority order. It is interesting that while the *policer* is flagging *packets* as in-profile or out-of-profile on a per-flow basis, the *WFQ scheduler* is marking *queues* as being in-profile or out-of-profile.

B. Experiment 1

1) Purpose and execution: The goals of this experiment were to (i) determine the router buffer size, (ii) determine the default mode used in the routers for link capacity (rate) sharing (between the two options of strict partitioning and work-conserving), and (iii) compare these two modes. Correspondingly, three scenarios were tested with different router configurations. To control rate and buffer allocations, the router software required the configuration of a virtual queue on the egress interface, even if it was just a single queue to which all flows were directed. In scenario 1, by modifying the buffer allocation for the virtual queue, router buffer size was determined. In scenario 2, by modifying the rate allocation, the default mode for capacity sharing was determined. Finally, in scenario 3, the router was explicitly configured to operate in the two different modes for comparison.

As per our execution methodology, the first step was to plan applications. For the first two scenarios, we planned to use two UDP flows created by the `nuttcp` application, and a "ping" flow to send repeated echo-request messages and receive responses. The purpose of the ping flow was to measure round-trip delays, and the UDP flows were used to fill up the router buffer. Only one UDP flow was required for the third scenario. Hosts `W1` and `E2` were used to generate the two UDP flows, both of which were destined to host `E1`. Different hosts were used to achieve high transfer rates. The ping flow sent messages from host `WA` to host `EA`. Therefore, contention for buffer and bandwidth resources occurred on the link from router `WR` to router `ER`.

Our next step was to configure the routers. A single virtual queue was configured on the output interface from `WR` to `ER`, and all application flows were directed to this queue. In scenario 1, the whole link capacity was assigned to the virtual queue, but the buffer allocation was changed from 20% to 100%. In scenario 2, the assigned rate was varied from 1% to 100%, while the buffer allocation was set to 100%, and in scenario 3, the rate and buffer allocations were set to 20%, and the capacity sharing mode was explicitly configured.

Next, we executed the experiments corresponding to the scenarios. For the first two scenarios, each `nuttcp` application was initiated with the sending rate set to 7 Gbps, resulting in a total incoming rate of 14 Gbps in order to fill up the buffer of the 10 Gbps `WR`-to-`ER` interface. Due to the resulting packet losses, `nuttcp` at the receiving host `E1` reported rates of approximately 5 Gbps for each UDP flow. In scenario 3, the sending rate of the single UDP flow was set to 3 Gbps. This was sufficient given the 20% rate allocation to the configured virtual queue on the `WR`-to-`ER` link in this scenario. In all three scenarios, the UDP flows and ping flow were run for 60 seconds.

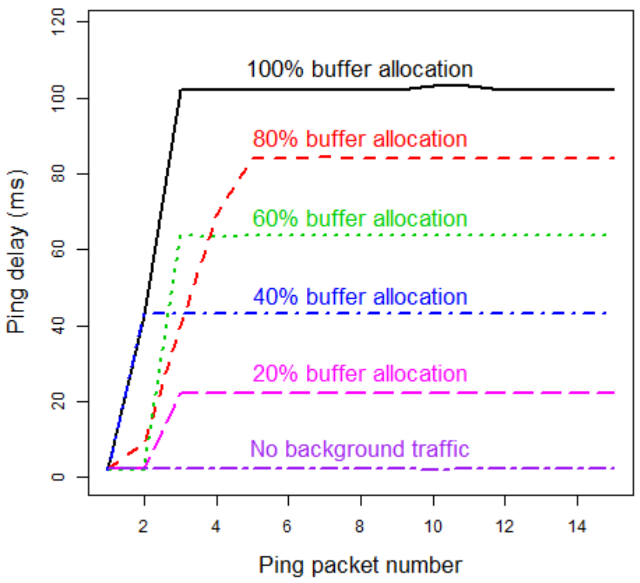


Figure 3. Experiment 1 scenario 1 results: Ping delay for different buffer allocations (rate allocation was 100%)

Finally, for the first and third scenarios, round-trip time (delay) measurements were obtained from the ping application on the `WA` host. For the second scenario, router counters for outgoing packets on the `WR`-to-`ER` link were read in order to find the number of packets transmitted within 60 seconds under different rate allocations.

2) Results and discussion:

Router buffer size: The ping packet delay measured in scenario 1 is plotted against the ping packet number, which is effectively the same as time, in Figure 3. With increasing time, the ping delay increases gradually because the `nuttcp` UDP packets start filling the buffer partition allocated to the virtual queue on the `WR`-to-`ER` interface. The minimum ping delay (2.1 ms) was observed when there were no UDP flows, i.e., there was no background traffic. The maximum delay (102 ms) was observed when the buffer allocation for the virtual queue was 100%.

In the various plots of Figure 3, the buffer allocations for the virtual queue are indicated. When the buffer allocation was limited to 20%, the delay was only 22.2 ms, while when the buffer allocation was set to 100%, the ping delay was higher because the whole buffer had filled up. Recall that the aggregate arrival rate of packets destined for the `WR`-to-`ER` link was 14 Gbps, while the outgoing link rate was only 10 Gbps.

Based on these observations, the buffer size for the `WR`-to-`ER` egress interface can be computed as follows:

$$10\ Gbps \times (102 - 2.1)\ ms = 125\ MB \qquad (1)$$

Default mode for link capacity sharing: From the experiments conducted in Scenario 2, the router counters for the `WR`-

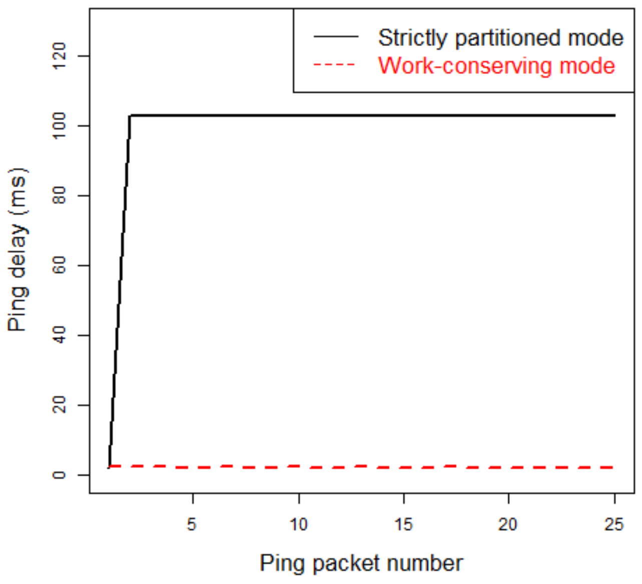


Figure 4. Experiment 1 scenario 3: Results comparing the two rate sharing modes (rate and buffer allocation was 20%)

to-ER were recorded, and are shown in Table I. The reported packets were almost the same for all values of the link capacity allocation. Recall that the buffer allocation was set to 100% for this scenario. In other words, even if only a 1% rate was assigned to the virtual queue in which packets from all three flows were held, the virtual queue was served at 100% capacity. This result verifies that the default mode of operation for the tested router is the work-conserving mode.

Comparison of the two rate sharing modes: Two rate sharing strategies, strictly partitioned and work conserving, were compared in Scenario 3. Figure 4 shows the ping delay results under these two configurations. In the strictly partitioned configuration, ping delays built up to 102 ms. Recall that for scenario 3, the virtual queue rate and buffer allocations were set to 20%, which was confirmed as follows:

$$R = \frac{125\ MB \times 0.2}{(102 - 2.1)\ ms} = 2\ Gbps \tag{2}$$

Under the work-conserving configuration, ping delay was only 2.1 ms (the round-trip time with no background traffic). Recall that the UDP flow sending rate was 3 Gbps in this scenario, while the rate allocation was only 2 Gbps. Yet there was no queue buildup in the buffer, which means the egress interface was served at a rate greater than 3 Gbps. Thus, in the work-conserving mode, virtual queues that have packets are served with excess capacity, if any.

C. Experiment 2

1) Purpose and execution: The goals of this experiment were to (i) determine whether α flows have adverse effects on real-time flows, and (ii) determine whether a scheduling-only no-policing solution of α-flow isolation to a separate virtual queue is sufficient to meet the first goal of keeping non-α flow delay/jitter low.

The first step was to plan a set of applications. We decided to use four `nuttcp` TCP flows and a ping flow. The TCP version used was H-TCP [18] because it is the recommended option to create high-speed (α) flows [19]. Two of the TCP flows carried data from host `E2` toward host `W1`, while the other two TCP flows were from `E1` to `W1`. The ping flow was from `EA` to `W1`. Therefore, in this experiment, contention for buffer and bandwidth resources occurred on the link from router `WR` to host `W1`. Although the high-performance host `W1` was the common receiver for all flows, there was no contention for CPU resources at `W1` because the operating system automatically scheduled the five receiving processes to different cores.

The second step was to configure the routers. For comparison purposes, this experiment required two configurations: (i) `1-queue`: a single virtual queue was defined on the egress interface from `WR` to `W1`, and all flows were directed to this queue, and (ii) `2-queues`: two virtual queues (α queue and β queue) were configured on the egress interface from `WR` to `W1`, and WFQ scheduling was enabled with the following rate (and buffer) allocations: 95% for α queue and 5% for β queue. The priority levels of the α and β virtual queues were set to medium-high and medium-low, respectively. In the 2-queues configuration, two additional steps were required. A firewall filter was created in router `WR` to identify packets from TCP (α) flows using their source and destination IP addresses. A class-of-service configuration command was used to classify these packets as belonging to the α class and to direct packets from these flows to the α queue on the egress interface from `WR` to `W1`. By default, all other packets were directed to the β queue, which means that packets from the ping flow were sent to the β queue.

In the third step, the applications were executed as follows. The four TCP flow execution intervals were: (0, 200), (20, 160), (40, 140), and (60, 120), respectively, while the ping flow was executed from 0 to 200 seconds.

Finally, throughput measurements as reported by each `nuttcp` sender were collected, as were the delays reported by the ping application.

TABLE I. EXPERIMENT 1 SCENARIO 2: PACKET COUNTER VALUES OBSERVED AT ROUTER `WR` FOR ITS `WR`-TO-`ER` INTERFACE

Link rate allocation	1%	20%	40%	60%	80%	100%
Number of packets transmitted on the `WR`-to-`ER` link	51924370	51536097	52755553	52669911	52786301	52637553

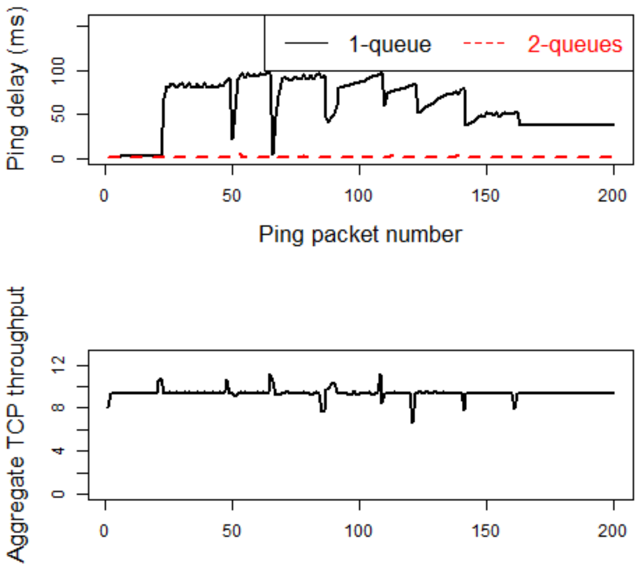


Figure 5. Experiment 2: Top graph shows the delays experienced in the ping flow under 1-queue and 2-queues configurations; bottom graph shows the aggregate TCP flow throughput

2) Results and discussion: The top graph in Figure 5 illustrates that the scheduling-only no-policing solution of configuring two virtual queues on the shared egress interface and separating out the α flows into their own virtual queue leads to reduced packet delay/jitter for the β flow. In the 1-queue configuration, the mean ping delay was 60.4 ms, and the standard deviation was 29.3 ms, while in the 2-queues configuration, the mean ping delay was only 2.3 ms, with a standard deviation of 0.3 ms.

In the 2-queues case, since the rate of the ping flow was much lower than the 5% allocated rate for the β queue, the β queue was in-profile, and hence the ping-application packets were served immediately without incurring any queueing delays.

The bottom graph in Figure 5 shows the aggregate throughput of the four TCP flows. A comparison of this throughput graph with the top ping-delay graph shows the following: (i) when the aggregate TCP throughput increased from 9.4 Gbps to 10.7 Gbps at time 22, and the ping delay increased from 3 ms to 82 ms. The `nuttcp` application reports average throughput on a per-sec basis. Therefore, while the total instantaneous throughput cannot exceed 10 Gbps (link rate), the sum of the per-sec average throughput values for the four TCP flows sometimes exceeds 10 Gbps, (ii) when the aggregate TCP throughput dropped from 10.6 Gbps to 9.3 Gbps at time 49, the ping delay dropped from 92 ms to 22 ms, correspondingly, and (ii) throughput drops at 85, 121, 141, and 161 sec coincided with ping-delay drops.

TABLE II. EXPERIMENT 3: α-FLOW THROUGHPUT UNDER DIFFERENT BACKGROUND LOADS (UDP RATE) AND QOS CONFIGURATIONS

UDP rate (Gbps)	α-flow throughput (Gbps)			
	Percentages for 2-queues (α, β) and 3-queues (α, β, *SS*) configurations			
	(50,50)	(49,50,1)	(30,50,20)	(10,30,60)
0	9.12	9.09	9.07	9.12
0.5	8.92	6.62	6.06	6.83
1	8.43	5.22	5	2.12
1.5	7.94	3.78	3.67	2.82
2	7.44	2.7	1.93	0.92
2.5	6.95	0.33	1.38	0.69
3	6.46	0.34	0.38	0.61

D. Experiment 3

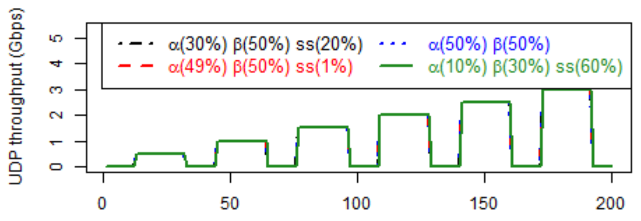
1) Purpose and execution: The goals of this experiment were to (i) compare a 2-queues configuration (scheduling-only, no-policing) with a 3-queues configuration (scheduling and policing), and (ii) compare multiple 3-queues configurations with different parameter settings.

As per our execution methodology, the first step was to plan applications. To study the behavior of the QoS mechanisms, one `nuttcp` TCP flow and one `nuttcp` UDP flow (background traffic) were planned. The UDP flow carried data from host E2 toward host W1, while the TCP flow was from E1 to W1. Contention for buffer and bandwidth resources occurred on the link from router WR to host W1.

In the second step, the router `WR` was configured with the following QoS mechanisms. The 2-queues configuration was the same as in experiment 2 (no-policing), except that both queues were given equal weight in sharing the rate and buffer (50% each). For the 3-queues configurations, the allocations for the three queues (α, β, and *SS*) to which in-profile TCP-flow packets, UDP and ping packets, and out-of-profile TCP-flow packets, were directed, respectively, are shown in Table II. The priority levels of these three virtual queues were medium-high, medium-low, and low respectively. The policer was configured to direct in-profile TCP-flow packets ($\leq$ 1 Gbps and burst-size $\leq$ 31 KB) to the α queue, and out-of-profile packets to the *SS* queue.

In the third step, experiment execution, the UDP flow rate was varied from 0 Gbps to 3 Gbps in a particular on-off pattern as shown in the top graph of Figure 6, and the TCP flow was executed for the whole 200 sec. Finally, the same performance metrics were collected as in experiment 2.

2) Results and discussion: Figure 6 shows the TCP throughput under the four configurations (one 2-queues and three 3-queues) for different rates of the background UDP flow. When the UDP flow rate was non-zero, since some of the plots overlap, we have summarized the mean TCP-flow throughput in Table II. When there was no background UDP traffic, the throughput of the TCP flow was around 9.1



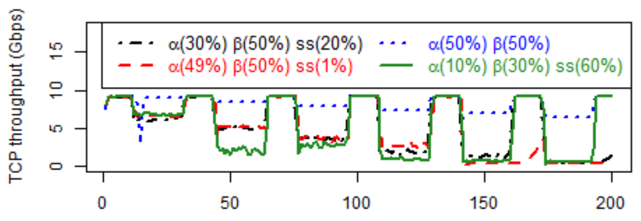


Figure 6. Experiment 3: The x-axis is time measured in seconds; the top graph shows the on-off mode in which the UDP rate was varied; the lower graph shows the TCP flow throughput under the four configurations.

Gbps for all four configurations as seen in the first row of Table II. As the background traffic load was increased, the throughput of the TCP flow in all the 3-queues configurations dropped more rapidly than in the 2-queues configuration, e.g., when the background UDP-flow rate was 3 Gbps, the TCP throughput was in the 300-610 Mbps range for the 3-queues configurations, while the TCP throughput was 6.5 Gbps for the 2-queues scenario (see last row of Table II).

In addition to explaining the first and last rows of Table II, we provide an explanation for the drop in TCP-flow throughput in the last column of the row corresponding to UDP rate of 1 Gbps, which highlights the importance of choosing the WFQ allocations carefully.

Explanation for the first row of Table II: The explanation for the TCP-flow throughput when there was no background traffic is straightforward in the 2-queues configuration. As there were no packets to be served from the β queue and the transmitter was operating in a working-conserving manner, the β queue's 50% allocation was used instead to serve the α queue, and correspondingly the TCP flow enjoyed the full link capacity.

The explanation for the TCP-flow throughput values observed in the 3-queues configurations requires an understanding of the packet arrival pattern to the policer (see Figure 2) and the rate at which packets leave the policer. When TCP-flow throughput was almost the line rate (over 9 Gbps), then the rate at which in-profile packets left the policer was almost constant at 1 Gbps. This is because the token generation rate was 1 Gbps and packet inter-arrival times were too short for a significant collection of tokens in the bucket. Therefore, in an almost periodic manner, every tenth packet of the TCP flow was marked as being in-profile and sent to the α queue and the remaining 9 packets were classified as out-of-profile and sent to the *SS* queue. Given that in all the 3-queues configurations, the α queue was assigned at least 10% of the link rate/buffer space, the WFQ scheduler determined that the α queue was in-profile, and the PQ scheduler systematically served 1 packet from the α queue followed by 9 packets from the *SS* queue thus preserving the sequence of the TCP-flow packets. In the (49,50,1) configuration, 9 packets were served out of the SS queue in sequence even though the queue was out-of-profile after the first packet was served. This is because there were no packets in the β queue and none in the α queue given the policer's almost-periodic direction of 1-in-10 packets to this queue. Since no packets were out-of-sequence or lost, the TCP-flow throughput remained high at above 9 Gbps in all the 3-queues configurations.

Explanation for the last row of Table II: When there was background `nuttcp` UDP traffic at 3 Gbps, in the 2-queues configuration, it is easy to understand that the `nuttcp` TCP flow was able to use up most of the remaining bandwidth, which is the line rate minus the rate of background `nuttcp` UDP flow, and hence the TCP-flow throughput was about 6.5 Gbps.

The explanation for the low `nuttcp` TCP throughput in the 3-queues configurations is that the opposite of the systematic behavior explained above for the first row occurred here. When the incoming packet rate to the policer was lower than the line rate, the token bucket had an opportunity to collect a few tokens. Therefore, when TCP-flow packets arrived at the policer, a burst of them was classified as in-profile (since for every token present in the bucket, one packet is regarded as being in-profile), and sent to the α queue. These were served in sequence, but because the transmitter had to serve the β queue (for the UDP flow), the pattern in which the policer sent packets to the α queue and *SS* queue is unpredictable and involved bursts. This resulted in TCP segments arriving out-of-sequence at the receiver (as confirmed with `tcpdump` and `tcptrace` analyses presented in the next section). Out-of-sequence arrivals trigger TCP's Fast retransmit/Fast recovery algorithm, which causes the sender's congestion window to halve resulting in lower throughput.

Explanation for the last-column entry in the row corresponding to 1 Gbps in Table II: The TCP-flow throughput dropped much faster from 6+ Gbps to 2.12 Gbps when UDP rate increased from 0.5 to 1 Gbps in the (10,30,60) 3-queues configuration than in the other two 3-queues configurations. This is explained using the above-stated reasoning that when the TCP-flow packets do not arrive at close to the line rate, the inter-packet arrival gaps allow the token bucket to collect a few tokens, making the policer send bursts of packets to the α queue. In this (10,30,60) configuration, after serving only one packet from each burst, the WFQ scheduler found the α queue to be out-of-profile since its allocation was only 10% or equivalently 1 Gbps. This led to a greater number of out-of-sequence arrivals at the TCP receiver than in the other two 3-queues configurations, and hence lower throughput.

TABLE III. EXPERIMENT 4: QOS CONFIGURATIONS; OOP: OUT-OF-PROFILE

Configuration	Policing	WFQ allocation `2-queues`:(α,β) `3-queues`:(α,β,`SS`)	WRED
`2-queues`	None	(60,40)	NA
`3-queues + policing1`	OOP to `SS` queue	(59,40,1)	NA
`3-queues + policing2`	OOP to `SS` queue	(20,40,40)	NA
`2-queues + policing + WRED`	WRED	(60,40)	Drop prob. = queue occ.

In *summary*, the higher the background traffic load, the lower the `nuttcp` TCP-flow packet arrival rate to the policer, the larger the inter-arrival gaps, the higher the number of collected tokens in the bucket, and the larger the number of in-profile packets directed to the α queue. If the WFQ allocation to the α queue is insufficient to serve in-profile bursts, packets from the α queue and *SS* queue will be intermingled resulting in out-of-sequence packets at the receiver. This fine point notwithstanding, the option of directing out-of-profile packets from the policer to a separate queue appears to be detrimental to α-flow throughput. We conclude that the second goal of high α-flow throughput cannot be met with this policing approach. In the next experiment, a different mechanism for dealing with out-of-profile packets was tested.

E. Experiment 4

1) Purpose and execution: The goal of this experiment was to compare the approach of applying WRED to out-of-profile packets rather than redirecting these packets to a scavenger-service queue as in experiment 3. The planned applications were the same as in experiment 3, i.e., to generate one `nuttcp` TCP flow and one `nuttcp` UDP flow.

The next step was router configuration. Four configurations are compared as shown in Table III. In the fourth option, Out-of-Profile (OOP) packets are dropped probabilistically at the same rate as the fraction of α-queue occupancy. In other words, if the α queue has 50% occupancy, then 50% of the OOP packets are dropped on average. The policing rate and burst size settings were the same as in Experiment 3.

Both the TCP and UDP flows were executed for 200 sec, but unlike in experiment 3, the rate of the UDP flow was maintained unchanged at 3 Gbps for the whole time period. Finally, in addition to the previously used methods of obtaining throughput reports from `nuttcp`, two packet analysis tools, `tcpdump` and `tcptrace`, were used to determine the number of out-of-sequence packets at the receiver. Additionally, to find the number of lost packets, a counter was read at router `WR` for the `WR`-to-`W1` link before and after each application run.

2) Results and discussion: The lower graph in Figure 7 and Table IV show that the TCP-flow throughput is highest in the

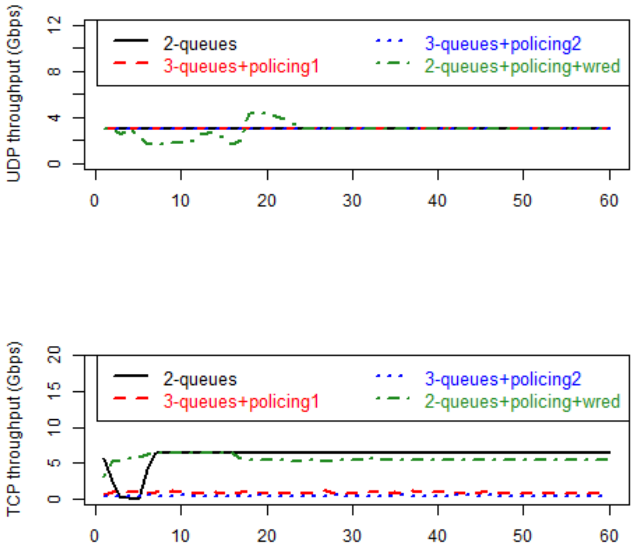


Figure 7. Experiment 4: The x-axis is time measured in seconds; the top graph shows the on-off mode in which the UDP rate was varied; the lower graph shows the TCP flow throughput under the four configurations.

2-queues (no-policing) scenario, with the WRED option close behind. The policing with WRED option performs much better than the options in which out-of-profile (OOP) packets are directed to an *SS* queue. In the WRED-enabled configuration, the TCP flow experiences a small rate of random packet loss, as shown in Table IV, while in 3-queues configurations, there were much higher numbers of out-of-sequence packets. The out-of-sequence packets in the WRED-enabled configuration result from the 15 lost packets, and are not independent events.

Surprisingly, even though the number of out-of-sequence packets was larger for the `3-queues+policing1` configuration, the throughput was higher in that configuration. This implies that fewer number of the out-of-sequence packets caused triple-duplicate ACKs in the first case. But this pattern is likely to change for repeated executions of the experiment.

Finally, Figure 7 shows that in the `2-queues` (no-policing) configuration, there was degradation of throughput soon after the flow started. Also, Table IV shows a loss of 5050 packets (the 4076 out-of-sequence packets were related to these losses). Using `tcptrace`, we found that these losses occurred at the start of the transfer. This is explained by the aggressive growth of the congestion window (`cwnd`) in H-TCP, which uses a short throughput probing phase at the start. During the 1st second, the throughput of the TCP flow averaged 5.7 Gbps. The 5050 lost packets occurred in the 2^{nd} second. These losses occurred in the `WR` router buffer on its egress link from `WR` to `W1`. If H-TCP increased its `cwnd` to a large enough value to send packets at an instantaneous rate higher than 7 Gbps, then given the presence of the UDP flow at 3 Gbps, the α queue would fill up. From experiment 1, we determined that

TABLE IV. EXPERIMENT 4: NUMBER OF OUT-OF-SEQUENCE PACKETS AND LOST PACKETS FOR DIFFERENT QOS SETTINGS

Measure	2-queues	3-queues+ policing1	3-queues+ policing2	2-queues+ policing+wred
Average throughput	6 Gbps	0.92 Gbps	0.47 Gbps	5.6 Gbps
Num. of out-of-sequence packets at the receiver	4076	8812	7199	15
Num. of lost packets at the WR-to-W1 router link	5050	0	0	15

the particular router used as WR has a 125 MB buffer. Since the buffer is shared between the α and β queues in a strictly partitioned mode with the 60-40 allocation, the α queue has 75 MB, which means that if the H-TCP sender exceeds the 7 Gbps rate by even 600 Mbps, the α queue will fill up within a second. Inspite of this initial packet loss, the 2-queues no-policing configuration achieves the highest throughput. In the next experiment, we consider the question of whether the use of policing and WRED has a fairness advantage when multiple α flows share a queue.

F. Experiment 5

1) Purpose and execution: The goal of this experiment was to understand how two α flows compete for bandwidth under different 2-queues configurations: without policing (2-queues), and with policing and WRED (2-queues+policing+WRED). In a first scenario, the α flows had similar round-trip times (RTTs), while in a second scenario, the RTTs differed significantly. We expected a fairness advantage for the policing/WRED scheme, but found the opposite. This is because neither of the two policed α flows honored their assigned rates, and while under the no-policing scheme the TCP flows adjusted their sending rates and had no packet losses, the deliberate packet losses in the policing/WRED scheme lowered throughput and resulted in a lower fairness. Thus, the no-policing configuration outperforms the configuration with policing and WRED from both throughput and fairness considerations when neither flow honors the policed rate.

The first step was to choose applications. Two nuttcp TCP flows were planned. The first TCP flow (TCP1) was from host E2 to host W1, and the second TCP flow (TCP2) was from host E1 to host W1. The RTTs were similar but not exactly the same. The RTT was 1.98 ms on the E2-to-W1 path and 2.23 ms on the E1-to-W1 path, because the latter path passes through an additional router, ER.

The router configurations were as follows. In the 2-queues configuration, packets from both TCP flows were directed to an α queue, with the rate and buffer allocations set to (60,40) for the α and β queues, respectively. In the 2-queues+policing+WRED configuration, the policing rate/burst size settings were the same as in Experiment 3, and Out-of-Profile (OOP) packets were dropped probabilistically with the same settings as in Experiment 4 ([0,100] drop

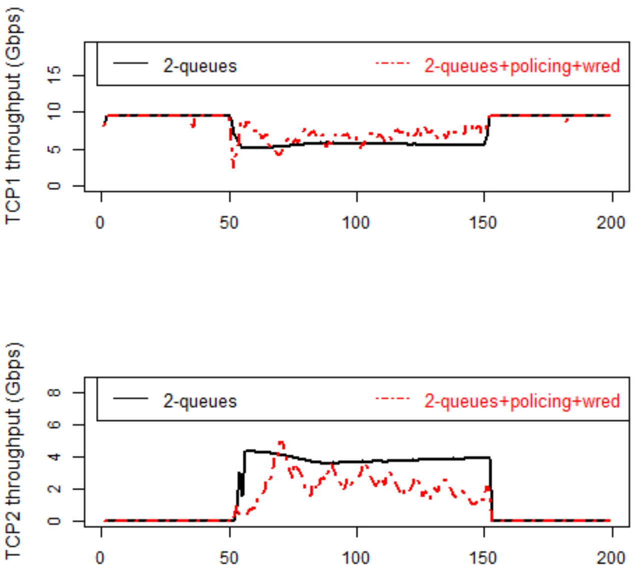


Figure 8. Experiment 5: Throughput of two TCP flows under two QoS configurations (similar RTTs)

probability corresponding to [0,100] buffer occupancy.

TCP1 and TCP2 execution intervals were (0, 200) and (51, 151), respectively. In the different-RTTs scenario, the RTT of TCP2 was increased by 50 ms using tc. Finally, throughput and retransmission data were collected every second by nuttcp at the senders.

2) Results and discussion: Experimental results are presented for the similar-RTT and different-RTTs scenarios.

Similar-RTT scenario:

Figure 8 shows the throughput of the two TCP flows when they compete for the bandwidth and buffer resources of the α queue. In the 2-queues configuration, the throughput of TCP1 was approximately 9.4 Gbps for the first 50 seconds, but dropped to 7.1 Gbps at $t = 51$, since TCP2 was initiated then. In the 52^{nd} second, both flows suffered packet losses, with TCP1 requiring 2418 retransmissions and TCP2 requiring 3818 retransmissions. Since the sum of the rates of the flows exceeded 10 Gbps, it caused losses and retransmissions in the 52^{nd} sec. After the 52^{nd} second, there were no retrans-

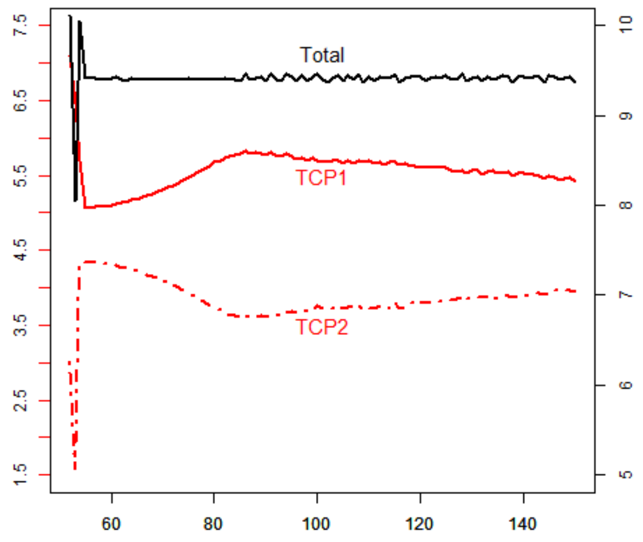


Figure 9. Experiment 5: Throughput of two TCP flows, and their total throughput in the `2-queues` configuration (similar RTTs)

missions on either flow. The per-second throughput recorded by `nuttcp`, from $t = 51$ to $t = 151$ during which both TCP flows were active, is shown in Figure 9. As the buffer filled up and queueing delays increased, TCP acknowledgments (ACKs) would have been delayed causing RTT for `TCP1` to increase. This decreased the effective sending rate (`cwnd`/RTT). No losses occurred in the rest of the experiment because as sending rates increased in one or both flows, the buffer filled up delaying packets, and hence increasing RTT, which in turn caused the sending rate to drop thus reducing buffer buildups. This oscillatory behavior can be observed in the throughput sum plot of Figure 9. The higher rate of `TCP1` could be because of the slightly lower RTT for this flow when compared to that of TCP2.

TABLE V. EXPERIMENT 5: RETRANSMISSIONS AND THROUGHPUT OF 2 TCP FLOWS FOR THE POLICING/WRED CONFIGURATION (SIMILAR RTTs)

Time (s)	TCP	Retrans-missions	Throughput (Gbps)		
			Min	Median	Max
51 - 53	TCP1	227	2.56	4.84	6.31
	TCP2	32	0.46	0.5	1.03
54 - 69	TCP1	14	4.35	7.37	8.98
	TCP2	0	0.47	1.78	4.98
70 - 151	TCP1	65	4.26	6.91	8.47
	TCP2	3	1.02	2.26	4.26

Next, consider the throughput values of `TCP1` and `TCP2` in the `2-queues+policing+WRED` configuration shown in Figure 8. From $t = 51$, `TCP1` suffered losses and its throughput dropped steadily until it reached 4.35 Gbps, while `TCP2` throughput kept increasing until it reached 4.98 Gbps at $t = 69$. The reason why `TCP1` throughput dropped is because of the policing limit of 1 Gbps. Packets exceeding this rate were marked as out-of-profile. Since `TCP1` rate was 9.4 Gbps at $t = 50$ just before `TCP2` was started, its sending rate was well above the policing rate of 1 Gbps. Subsequent to reaching this almost balanced throughput level at $t = 69$, losses, and hence retransmissions, were observed on both flows, but there were more losses in `TCP1` (see Table V) because its rate was higher.

The key difference between the `2-queues` and `2-queues+policing+WRED` configurations is that there were no losses in the former configuration after $t = 53$, while in the latter configuration both flows kept experiencing packet losses. This is because in the second configuration, as both flows exceeded the policing limit of 1 Gbps, a few packets were marked as out-of-profile in both flows. Recall that under WRED packets are dropped probabilistically at a rate equal to buffer occupancy, and since the buffer will sometimes have packets, losses are inevitable in the `2-queues+policing+WRED` configuration. When losses occurred under the `2-queues+policing+WRED` configuration, the slight edge in RTT for `TCP1` may account for its higher throughput when compared to `TCP2`. `TCP1` maintained an average rate of 6.86 Gbps from $t = 70$ to $t = 151$ when `TCP2` was terminated, at which point `TCP1` recovered its rate to 9.4 Gbps. The `TCP2` average throughput from $t = 70$ to $t = 151$ was smaller at 2.35 Gbps. A loss detected with triple duplicate ACKs results in a halving of `cwnd`, which is equivalent to halving the sending rate. `TCP1` operated in a higher range of `cwnd` values when compared to `TCP2`.

Using Jain's fairness index [20],

$$f(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \quad x_i \geq 0 \tag{3}$$

and average throughput values across the $t = 51$ to $t = 151$ time range, we computed the fairness values to be 0.97 and 0.8 for the `2-queues` and `2-queues+policing+WRED` configurations, respectively. This does not imply that the former is a more fair configuration; it is just that in this experiment, given that both TCP flows did not honor the policing limit, policing caused packet losses, and recovery from packet losses was slower for the longer-RTT path even if the RTT difference was small. Without policing, there were no deliberate packet drops in the `2-queues` configuration; instead the TCP senders self-regulated their sending rates. When the rates were high, buffer occupancy grew, but this caused RTT to increase, which, in turn, caused a lower sending rate.

In *summary*, this experiment showed that policing will result in decreased throughput for TCP based α flows when two or more such flows occur simultaneously. In Experiment 4, policing with WRED did not impact throughput significantly but there was only one TCP based α flow, unlike in this

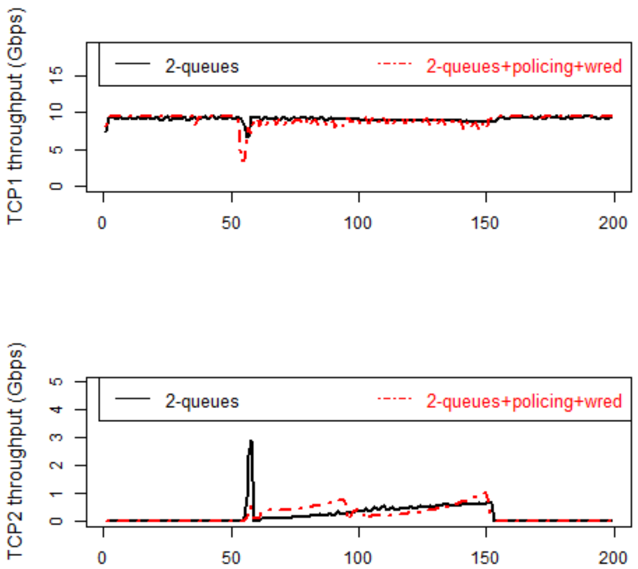


Figure 10. Experiment 5: Throughput of two TCP flows under two QoS configurations (different RTTs)

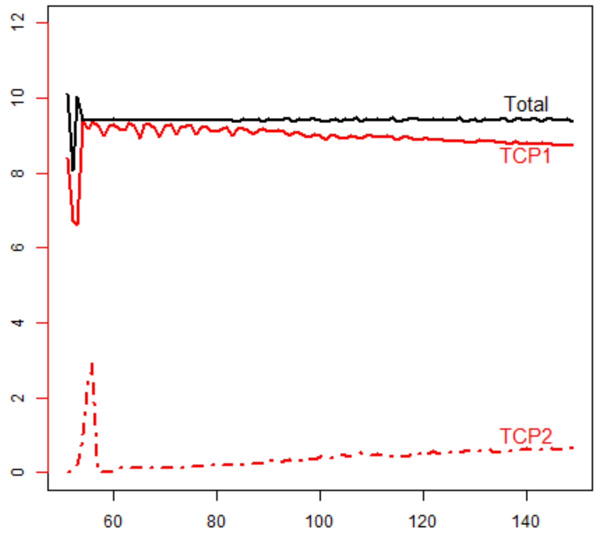


Figure 11. Experiment 5: Throughput of two TCP flows, and their total throughput in the `2-queues` configurations (different RTTs)

experiment.

Different RTTs:

Figure 10 shows the throughput of the two TCP flows with different RTTs. During the 100-second period when both TCP flows were active, the throughput of the two TCP flows and their total throughput are plotted in Figure 11. The throughput of `TCP1` dropped from 9.1 Gbps at $t = 50$ to 7.1 Gbps at $t = 51$, since `TCP2` was initiated at time 50. In the 57^{th} second, when `TCP2` built up its rate to 2.93 Gbps, which made the sum of the rates exceed 10 Gbps, both flows suffered packet losses, with `TCP1` requiring 3315 retransmissions and `TCP2` requiring 4118 retransmissions. After the 57^{th} second, there were no retransmissions on either flow. Since the RTT of `TCP2` was increased by 50 ms, it took 6 sec to reach the time instant when losses occurred unlike in the similar-RTT scenario in which both flows experienced losses in 2 sec. In the second after the losses, `TCP1` recovered its throughput back to 9.38 Gbps, while `TCP2` throughput decreased from 2.92 Gbps to 11 Mbps.

TABLE VI. EXPERIMENT 5: RETRANSMISSIONS AND THROUGHPUT OF 2 TCP FLOWS FOR THE POLICING-WRED CONFIGURATION (DIFFERENT RTTs)

Time (s)	TCP	Retrans-missions	Throughput (Gbps)		
			Min	Median	Max
51 - 58	TCP1	140	3.51	7.6	9.41
	TCP2	1	0.003	0.086	0.54
59 - 151	TCP1	107	7.61	8.81	9.35
	TCP2	4	0.056	0.42	0.98

Next, we repeated the experiments with the policing and WRED configuration. The retransmissions and throughput of the two TCP flows are shown in Table VI. `TCP1` experienced losses even after the initial set of losses unlike in the `2-queues` configuration. Consequently, TCP1's average throughput was lower in the `2-queues+policing+WRED` configuration (8.98 Gbps) than in the `2-queues` configuration (9.1 Gbps), while TCP2's average throughput was higher (0.43 Gbps vs 0.41 Gbps). Jain's fairness index value for throughput of the two TCP flows was comparable under the two configurations (0.546 and 0.551 under the `2-queues` and `2-queues+policing+WRED` configurations, respectively). The RTT difference was the dominant reason for the unfair treatment of `TCP2`, not the QoS configuration.

G. *Experiment 6*

1) Purpose and execution: The goals of this experiment were to (i) identify the impact of QoS provisioning under changing traffic conditions, and (ii) compare two versions of TCP: Reno and H-TCP. In the first part, we studied the effect of enabling QoS control, specifically, the `2-queues` configuration, on changing traffic patterns. For example, what is the impact of background traffic increasing to 3 Gbps when the β queue to which background traffic was directed was allocated only 20% of the rate/buffer capacity on a 10 Gbps link (based on previous traffic measurements). As α flows occur infrequently, most of the time, service quality for the background traffic would be unaffected, but if this surge in background traffic occurred within the duration of an α flow, there could potentially be higher losses and delays in the background traffic than if QoS mechanisms had not been enabled.

As mentioned in Section III-C, the TCP version used in our experiments was H-TCP, the recommended option for high-speed networks [19]. However, although computers dedicated for high-speed transfers are likely to be configured to use H-TCP, as the most widely used TCP version is still TCP Reno, we undertook a comparative experiment.

Three applications were planned for this experiment: one `nuttcp` TCP flow (from host `E1` to `W1`), one `nuttcp` UDP flow (from host `E2` to `W1`) and one ping flow (from host `EA` to `W1`). In the router configuration step, two queues were configured: a β queue for the background UDP flow and the ping flow, and an α queue for the TCP flow. The rate/buffer allocation (the same percentage was used for both resources) for the β queue was varied from 20% to 60% in 10% increments, and the allocations for the α queue were set correspondingly. The applications were executed as follows: UDP flow and ping flow in the time interval (0, 200), and the TCP flow in the interval (53, 153). Two rates were used for the UDP flow: 2 Gbps and 3 Gbps.

2) Results and discussion:

Goal 1: Table VII shows the UDP-flow loss rate and ping delay under different rate/buffer allocations for the β queue in the `2-queues` configuration. Before the TCP flow was initiated (the first 53 seconds) and after the TCP flow ends (the last 47 seconds), even if the rate of the UDP flow exceeded the allocated rate for the β queue (i.e., 20% allocation when the UDP-flow rate was 3 Gbps), the UDP flow experienced no losses, and the ping delay remained at around 2.26 ms, which implies that there was no buffer buildup in the β queue. This is because the transmitter was operating in work-conserving mode, which allowed it to serve packets from the β queue as the α queue was empty.

During the time interval (53-153) when the TCP flow was active, with a 20% rate/buffer allocation for the β queue, a 2 Gbps UDP flow suffered a 5% packet-loss rate, and the ping delay was 103 ms, which means the β queue was full. When the UDP-flow rate was increased to 3 Gbps, while the β-queue allocation was held at 20% (to model changing traffic conditions), the UDP-flow packet loss rate increased to about 39%, and the ping delay remained at 103 ms. Such a significant loss rate and increased packet delay would not have occurred had separate QoS classes not been created and the buffer not been divided. When the UDP-flow rate increased, the TCP-flow rate would have decreased as it would also have suffered losses. In the `2-queues` configuration, the TCP flow suffered no losses for both the combinations described above: 20% β queue allocation with 2 Gbps UDP-flow rate, and the 20%-3 Gbps combination. This is because the TCP flow was directed to the α queue, which had its own large (80%) buffer/rate allocation.

Goal 2: The numbers in Table VII show that there was no difference between H-TCP and Reno with regards to the impact of the TCP flow on the UDP and ping flows. Furthermore, Table VIII shows that the TCP flow enjoyed the same rate for most of its duration. When the background UDP-flow rate was 2 Gbps, the TCP-flow throughput was 7.45 Gbps, and when the UDP-flow rate was 3 Gbps, the TCP-flow throughput was correspondingly lower at 6.45 Gbps, irrespective of β-queue rate/buffer allocation. The only difference observed between Reno and H-TCP was in the TCP-flow's behavior in the first few seconds as shown in Table IX. Recall the TCP flow was started at $t = 53$. With Reno, the number of retransmissions that occurred in the early seconds drops as the β-queue buffer allocation was increased (and the α-queue size, to which the TCP flow was directed, correspondingly decreased). With smaller α-queue sizes, it appears that the TCP sender starts reducing its sending rate sooner, and hence there were fewer losses and retransmissions. We expected H-TCP to suffer more losses in the initial few seconds as it is more aggressive in increasing its sending window, but this was not observed. Both adjusted their sending rates and experienced no losses after the initial set of losses shown in Table IX.

TABLE VII. EXPERIMENT 6: UDP-FLOW LOSS RATE AND PING DELAY

	β queue rate and buffer allocation	UDP rate (Gbps)	UDP flow average packet loss rate before, during, and after the TCP flow			Average ping delay (ms) before, during, and after the TCP flow		
			t ∈ (0-52)	t ∈ (53-153)	t ∈ (154-200)	t ∈ (0-52)	t ∈ (53-153)	t ∈ (154-200)
Reno	20%	2	0	5.03%	0	2.25	103	2.26
	30%	2	0	0	0	2.3	2.25	2.25
	20%	3	0	39.33%	0	2.26	103	2.27
	30%	3	0	4.57%	0	2.27	104	2.31
	$\geq$ 30%	2 or 3	0	0	0	2.26	2.26	2.26
H-TCP	20%	2	0	5.3%	0	2.27	103	2.27
	30%	2	0	0	0	2.27	2.26	2.25
	20%	3	0	39.3%	0	2.26	103	2.27
	30%	3	0	4.67%	0	2.28	104	2.29
	$\geq$ 30%	2 or 3	0	0	0	2.26	2.27	2.27

TABLE VIII. EXPERIMENT 6: TCP-FLOW THROUGHPUT FOR MOST OF THE DURATION

Background (UDP) rate	TCP throughput	
	Reno	H-TCP
2 Gbps	7.45 Gbps	6.45 Gbps
3 Gbps	7.45 Gbps	6.45 Gbps

TABLE IX. EXPERIMENT 6: TCP-FLOW RETRANSMISSIONS IN ITS FIRST FEW SECONDS (THE FLOW WAS STARTED AT $t = 53$)

UDP-flow rate	β-queue rate/ buffer alloc.	Time	Number of retx pkts
Reno			
2 Gbps	30%	$t = 54$	6624
	40%	$t = 54$	5811
	50%	$t = 53$	4327
	60%	$t = 54$	2645
3 Gbps	30%	$t = 54$	7673
	40%	$t = 54$	6970
	50%	$t = 53$	5137
	60%	$t = 54$	3495
H-TCP			
2 Gbps	30%	$t = 54$	6008
	40%	$t = 54\&55$	4322 & 298
	50%	$t = 53$	3825
	60%	$t = 54$	3966
3 Gbps	30%	NA	0
	40%	$t = 54$	1423
	50%	NA	0
	60%	$t = 54$	3528

H. Experiment 7

1) Purpose and execution: As described in Section II, HNTES uses an offline approach by analyzing NetFlow reports of completed flows to determine source-destination addresses of α flows, and then uses these addresses to configure firewall filters in ingress routers of a provider's network to redirect packets of future α flows to traffic-engineered QoS-controlled paths. With this scheme, an α flow between a new source-destination pair will not be identified as such until its NetFlow reports are analyzed, which most likely will occur after the flow completes. Such unidentified α flows will be directed to the β queue in a `2-queues` configuration. Since in such a configuration, buffer resources are partitioned between the β queue and α queue, the purpose of this experiment was to study the impact of such unidentified α flows.

Three `nuttcp` flows were planned for this experiment: a UDP flow from `E2` to `W1`, TCP flow `TCP1` from `E2` to `W1`, and a second TCP flow `TCP2` from `E1` to `W1`. In addition, a ping flow was executed from from `EA` to `W1`. Two router configurations were used in this experiment: (i) `1-queue`: a single virtual queue was defined on the egress interface from `WR` to `W1`, and all flows were directed to this queue, and (ii) `2-queues`: two virtual queues (α queue and β queue) were

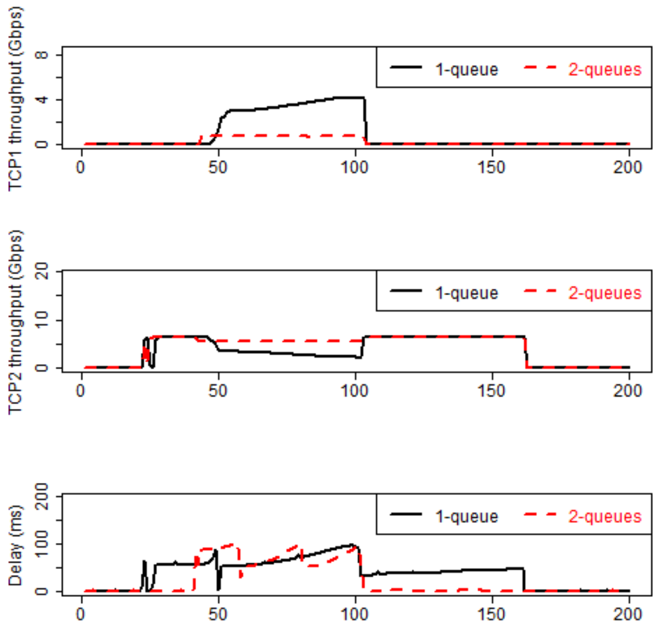


Figure 12. Experiment 7: The impact of an unidentified α flow with and without HNTES

configured on the egress interface from `WR` to `W1`, and WFQ scheduling was enabled with the following rate (and buffer) allocations: 60% for α queue and 40% for β queue.

The execution intervals of the flows, ping, UDP, `TCP1`, and `TCP2` were (0,200), (0, 200), (42, 101), and (22, 162), respectively. The rate of the UDP flow was set to 3 Gbps. We assumed `TCP1` to be the unidentified α flow, which was hence directed to the β queue, while TCP2 was assumed to be an α flow from a previously seen source-destination pair, and hence directed to the α queue. The ping and UDP flows were directed to the β queue. Measurements were collected from the `nuttcp` and ping applications.

2) Results and discussion: The throughput of the two TCP flows and the ping delays are shown in Figure 12. In the `1-queue` configuration, during the 60 seconds when both TCP flows were active, `TCP1` throughput kept increasing to 4.16 Gbps, while `TCP2` throughput kept decreasing from 6.5 Gbps to 2.26 Gbps. This is because the RTT was slightly lower for `TCP1` as discussed earlier.

In the `2-queues` configuration, `TCP1` throughput was only 1 Gbps. This is because the β queue allocation was 40% of the link rate/buffer, of which 3 Gbps was used by the UDP flow, and `TCP2` was actively consuming the 60% allocation of the α queue. The mean throughput of the new α flow (TCP1) in the `1-queue` case was 3.2 Gbps, while it was only 0.8 Gbps under the `2-queues` configuration. In other words, the presence of HNTES and the corresponding `2-queues` configuration had an adverse effect on the unidentified α flow, though as shown in our prior work, most α-flow generating source-destination pairs send repeated α flows [2].

Consider the impact of the unidentified α flow on the ping flow. In the `1-queue` configuration, the ping delay was

TABLE X. EXPERIMENT 7: TCP-FLOW RETRANSMISSIONS AND PING DELAYS

(sec, no. of `TCP1` retx)	(sec, no. of) `TCP2` retx)	(sec, ping delay (ms))
`1-queue configuration`		
NA	(23, 3267)	(24, 2.25)
(50, 955)	(50, 568)	(50, 4.7)
`2-queues configuration`		
NA	(22, 8074)	(22, 2.3)
(44, 1855)	(44, 0)	(44, 87.3)
(60, 7)	(60, 0)	(60, 30.2)
(82, 7)	(82, 0)	(83, 48.5)

around 2.3 ms until `TCP2` was initiated at $t = 22$, at which instant the ping delay surged up to 65.9 ms as seen in Figure 12 because of the buffer build-up from TCP2 packets. Since H-TCP is aggressive in increasing its sending rate, in its 2^{nd} second ($t = 23$), there were 3267 packet drops as shown in Table X. With all these losses, ping delay correspondingly dropped down to 2.25 ms at $t = 24$. However the delay quickly increased back to the 56 ms range peaking at 88.7 ms at $t = 49$. As shown in Table X, it took a few seconds after `TCP1` was initiated for both `TCP1` and `TCP2` to experience packet losses causing ping delay to drop back down to 4.7 ms at $t = 50$. Beyond this time instant, neither TCP flow suffered losses with both adjusting their sending rates based on received acknowledgments and ping delay peaked at 91.5 ms at $t = 101$ when `TCP1` ended. The ping delay dropped to 34 ms and increased to 47.9 ms at which point it dropped to 2.3 ms at $t = 162$ when `TCP2` ended.

In the `2-queues` configuration, the ping delay stayed around 2.3 ms even after `TCP2` was initiated as seen in Figure 12 (because `TCP2` was directed to a different queue), but increased to 87.3 ms when `TCP1` was initiated at $t = 43$ (since `TCP1` representing an unidentified α flow was directed to the same queue as the ping flow). `TCP2` suffered no losses after the initial losses of 8074 packets in its first second. On the other hand, `TCP1` suffered losses not only in its first second (1855 losses), but again at $t = 60$ and $t = 82$. During these seconds, ping delay dropped correspondingly from 103 ms at $t = 59$ to 30.2 ms at $t = 60$, and from 103 ms at $t = 82$ to 48.5 ms at $t = 83$. These results illustrate that the smaller buffer allocation for the β queue can have a negative effect on real-time flows when an unidentified α flow appears.

In summary, QoS partitioning does have negative effects when mismatched with traffic as shown in Experiment 6, and when α flows are undetected and hence handled by the partition set aside for β flows. Nevertheless, the benefits of QoS partitioning as illustrated in the first five experiments outweigh these costs.

IV. Conclusions and Future Work

To reduce the impact of large-sized, high-rate (α) transfers on real-time flows, a Hybrid Network Traffic Engineering System (HNTES) was proposed in earlier work. HNTES is an intra-domain solution that enables the automatic identification of α flows at a provider network's ingress routers, and redirects these flows to traffic-engineered QoS-controlled virtual circuits. The purpose of this work was to determine the best QoS mechanisms for the virtual circuits used in this application. Our findings are that a no-policing, two-queues (one for α flows and one for β flows) solution with weighted fair queueing and priority queueing is both sufficient and the best for this application. It allows for the dual goals of reduced delay/jitter in real-time flows, and high-throughput for the α flows, to be met.

We studied two types of policing schemes for handling out-of-profile packets: redirection to a (third) scavenger-service (SS) queue and Weighted Random Early Detection (WRED) in which out-of-profile packets are either dropped probabilistically according to some profile or held in the same queue as in-profile packets. The WRED scheme was better than the SS-queue scheme because the latter caused out-of-sequence arrivals at the receiver, which triggered TCP congestion control mechanisms that led to lower throughput. However, the no-policing solution was better than the policing/WRED solution because in this application flows are not likely to honor the circuit rates and therefore deliberate packet drops are inevitable in the policing/WRED solution causing lowered throughput. The negatives of partitioning rate/buffer space resources between two queues were studied. Our conclusions are that close network monitoring is required to dynamically adjust the rate/buffer space split between the two queues as traffic changes, and the probability of unidentified α flows should be reduced whenever possible to avoid these flows from becoming directed to the β queue.

As future work, we plan to develop theoretical and/or simulation models to characterize the impact of QoS provisioning schemes on TCP throughput.

V. Acknowledgment

The University of Virginia portion of this work was supported by the U.S. Department of Energy (DOE) grant DE-SC0002350, DE-SC0007341 and NSF grants OCI-1038058, OCI-1127340, and CNS-1116081. The ESnet portion of this work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. DOE under Contract No. DE-AC02- 05CH11231. This research used resources of the ESnet ANI Testbed, which is supported by the Office of Science of the U.S. DOE under contract DE-AC02-05CH11231, funded through the American Recovery and Reinvestment Act of 2009.

References

[1] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of nework traffic," ACM SIGCOMM Internet Measurement Workshop 2001, Nov. 2001, pp. 99-104.

[2] Z. Yan, C. Tracy, and M. Veeraraghavan, “A hybrid network traffic engineering system,” Proc. of IEEE 13th High Performance Switching and Routing (HPSR) 2012, Jun. 24-27, 2012, pp. 141-146.
[3] Esnet. Retrieved: 09.10.2013. [Online]. Available: http://www.es.net/
[4] GEANT. Retrieved: 09.10.2013. [Online]. Available: http://www.geant.net/
[5] JGN-X. Retrieved: 09.10.2013. [Online]. Available: http://www.jgn.nict.go.jp/english/
[6] On-Demand Secure Circuits and Advance Reservation System (OSCARS). Retrieved: 09.10.2013. [Online]. Available: http://www.es.net/services/virtual-circuits-oscars
[7] J. Spragins, “Asynchronous Transfer Mode: Solution for Broadband ISDN, Third Edition [New Books],” Jan./Feb. 1996, pp. 7.
[8] E. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol (RSVP),” RFC 2205, Sep. 1997.
[9] Z. Liu, M. Veeraraghavan, Z. Yan, C. Tracy, J. Tie, I. Foster, J. Dennis, J. Hick, Y. Li, and W. Yang, “On using virtual circuits for GridFTP transfers,” The International Conference for High Performance Computing, Networking, Storage and Analysis 2012 (SC 2012), Nov. 10-16, 2012, pp. 81:1-81:11.
[10] GridFTP. Retrieved: 09.10.2013. [Online]. Available: http://globus.org/toolkit/docs/3.2/gridftp/
[11] J. Kurose and K. Ross, “Computer networks: A top down approach featuring the internet,” Pearson Addison Wesley, 2010.
[12] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” IEEE/ACM Transactions on Networking, Aug. 1993, pp. 397-413.
[13] D. Lin and R. Morris, “Dynamics of random early detection,” ACM SIGCOMM Computer Communication Review, Oct. 1997, pp. 127-137.
[14] WRED. Retrieved: 09.10.2013. [Online]. Available: http://www.cisco.com/en/US/docs/ios/11_2/feature/guide/wred_gs.html
[15] R. Guérin, S. Kamat, V. Peris, and R. Rajan, “Scalable QoS provision through buffer management,” vol. 28, no. 4, Oct. 1998, pp. 29-40.
[16] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, “Approximate fairness through differential dropping,” ACM SIGCOMM Computer Communication Review, pp. 23–39, Apr. 2003.
[17] R. P. Vietzke. (2008, Aug.) Internet2 headroom practice. Retrieved: 09.10.2013. [Online]. Available: https://wiki.internet2.edu/confluence/download/attachments/17383/Internet2+Headroom+Practice+8-14-08.pdf
[18] D. Leith and R. Shorten, “H-TCP: TCP for high-speed and long-distance networks,” Proc. of PFLDnet, Feb. 16-17, 2004.
[19] Retrieved: 09.10.2013. [Online]. Available: http://fasterdata.es.net/host-tuning/linux/
[20] R. Jain, D. Chiu, and W. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,” *Technical Report TR-301, DEC Research*, Sep. 1984.