QoS aware TCP For Data Center Storage (QTCP)

Joyanta Biswas, Krishna Kant, Amitangshu Pal
Temple University, Philadelphia, PA 19122, USA

Dave Minturn
Intel Corp, Hillsboro, OR 97291

***Abstract*—Traditional TCP variants ensure fairness by equal distribution of the network resources amongst different applications and thus are unable to provide end to end QoS solution. With the introduction of increasingly speedier storage technologies that are primarily accessed over the data center network, there is a need for lightweight and QoS aware transport protocols. Given the ubiquity of TCP, the purpose of this paper is to introduce a lossless QoS aware TCP protocol called QTCP, that can enforce both the throughput and latency related QoS requirements (though currently not simultaneously). Like the well-established DCTCP, it leverages the buffer occupancy triggered Explicit Congestion Notification (ECN) in the switches to provide loss-free control of TCP transmission window. The proposed solution can co-exist with DCTCP and ensure RTT fairness while providing service differentiation. We compare QTCP against DCTCP and the deadline aware TCP (D^2TCP) proposal and show how it can support better and more flexible QoS aware treatment to applications. Although the motivation for designing QTCP comes from emerging high speed storage, it is not tied to storage in any way.**

***Index Terms*—Congestion control, TCP, Quality of Service, distributed storage**

I. Introduction and Motivation

Transmission control protocol (TCP) has long been a mainstay of Ethernet/IP based networks ubiquitously deployed in the entire Internet infrastructure. Consequently, numerous versions of TCP have been invented over the last few decades and many are currently in use, such as Reno, NewReno, Cubic, etc. [1] [2]. However, most of these variants are increasingly problematic within the confines of a data center network because of high data rates, undesirability of any packet loss, and the need for service differentiation. Network infrastructures especially designed for data center such as the Infiniband already provide the necessary features, including the very low-latency, hardware supported end to end protocol called RDMA (remote direct memory access) and QoS (by a switch-based mapping of service levels to virtual lanes). RDMA can be regarded as a counterpart of TCP, but achieves far lower latency due to hardware offload, lean lower layers, and minimal OS kernel involvement. However, with 100G+ speeds, Ethernet can be competitive even in high-end data centers, with TCP again being the most obvious choice for end to end data transfer. While TCP latency cannot match the native RDMA latency, RDMA implemented on top of TCP (known as iWarp) can achieve decent latency when assisted by suitable hardware offload and user space implementation [3], [4]. Low latency TCP implementations (including user-space implementations such as Solarflare [4] can help numerous traditional applications in enhancing their performance.

This research was supported by a grant from Intel Corp.

Many applications and basic services require quality of service (QoS) treatment. One important service that has motivated the work in this paper is the access to remote storage. Storage has always been accessed remotely regardless of whether it is distributed across individual servers (as in iSCSI), concentrated in a Storage Area Network (SAN) tower, or managed by one or more storage servers. For remote reads, the storage target is the transmitter and host is the receiver. For remote writes, the opposite is true. The end-to-end latency is clearly important for reads; it can also be important for writes if the write is released only after the target acknowledges it (regardless of whether the write actually makes it to the disk or held in a target buffer).

The emerging storage technologies can drive very high throughputs and provide low latency access. For example, a single Intel Optane SSD can drive about 25 Gb/sec and provides an access latency of about 10 μs [5]. Even regular SSDs (e.g., Samsung Evo370) can drive up to 35 Gb/sec bandwidth. The emerging persistent memory (PM) devices (e.g., Intel Optane PM) could support even higher bandwidths while doing very short (few cacheline) transfers and expect very low latencies due to the memory access model. Thus when such devices are accessed remotely over the network, the network latency becomes an important component of the overall access latency, particularly when the storage traffic can overwhelm some link in the path leading to the storage server. For example, a 4 KB transmission going through a couple of 100 Gb switches/routers could easily take at least a few us without any network queuing, and much more if the network is congested. Thus, network QoS becomes important, for example, to support a mix consisting of very low latency PM transfers, low latency 4KB Optane SSD transfers, and larger transfers to/from regular SSDs.

Although the IP layer supports the well-known DSCP (differentiated services code point) mechanism for QoS along a routable path, it has several limitations in the data center context. First, DSCP was developed for WAN and is based on packet loss characteristics. In data centers, packet loss is highly undesirable as it can cause the applications to hang or severly degrade their performance. Second, DSCP provides a hop-by-hop control wherein each switch along the path marks the packets according to the supplied policies. Instead, what is needed is an end-to-end mechanism that can be managed from the service endpoints. Third, the data center is not limited

to layer-2 switches, and routinely uses switches with layer-3 (or routing) capabilities. Thus, an Ethernet level QoS is itself insufficient. Also, while the data center bridging features (e.g., PFC – priority flow control, ETS – Enhanced transmission selection) [6] are well standardized and can be very useful, they are generally not implemented.

In view of the above, a practical approach is to implement a QoS aware transport layer that controls end to end data transfer rate based on the packet delays rather than drops. Data Center TCP (DCTCP) [7] has been defined to support such a functionality; however, DCTCP does not support service differentiation. The only other relevant TCP variant is the deadline driven TCP (D^{2}TCP) [8] that does provide service differentiation and will be discussed in section II.

In this paper, we present a QoS aware TCP, henceforth called QTCP, that like DCTCP and D^{2}TCP, is based on the loss-free ECN but is intended to support multiple QoS classes that may specify one of the following two goals under congestion: (a) Divide the bottleneck link bandwidth in specified proportions, or (b) Achieve specified end-to-end network latency objectives for the higher QoS classes. (Obviously, there must be at least one best-effort class with sufficient bandwidth that can be squeezed to achieve the latency objectives for others, and the latency objectives must be loose enough to be realizable.) Combinations of (a) and (b) are envisioned but beyond the scope of this paper. We show that QTCP is much more flexible than D^{2}TCP and can do a much better job of achieving the specified objectives.

Although the protocol is not tied to storage in any way and can be useful for many other purposes, the identification of application classes with respect to the bottleneck bandwidth subdivision or sensible latency targets could be challenging in general. In particular, for throughput control, we assume that the bottleneck bandwidth is either known from historical data or can be determined through continuous active monitoring.

The rest of the paper is organized as follows. Section II explores the background of different data center TCP mechanisms along with the motivation behind the work. Section III discusses our proposed QoS aware window modulation scheme. Analytical modeling of QTCP is explored in section IV. The evaluation results are covered in Section V. Section VI summarizes the related works. The paper is concluded in Section VII.

II. TCP for Data Center Use

To avoid the highly undesired packet losses in a data center network, the congestion must be triggered when the queue length at a switch along the path or at the receiver exceeds some value K that is substantially less than the actual buffer size. This feedback can occur through the standard ECN (explicit congestion notification) mechanism which operates as follows: The switches mark the congestion encountered (CE) bit in a packet when a queue length threshold K is exceeded. When the endpoint receives the packet, it sends the ECE (ECN-echo) message back to the sender which reduces its window. From latency perspective also, K needs to be sufficiently small for a data center environment regardless of the available buffer size. Also, in order to limit the tail latency, it is more appropriate for use tail drop (i.e., all packets that exceed the threshold K) instead of RED (random early drop) beyond K.

Both Data-center TCP (DCTCP) and Deadline-Aware Datacenter TCP (D^{2}TCP) [8] use an exponentially smoothed version of the ECN feedback to modulate the congestion window (cwnd) for reducing the amount of traffic into the network. The underlying control parameter is the fraction of acknowledgements in a window that arrive with the ECE bit set. Consider I competing TCP connections (or flows). Let $f_i(n)$ denote this fraction for ith flow during its nth window. Then we can obtain an exponentially smoothed version of this quantity over successive windows, popularly denoted as α_i, as follows:

$$\alpha_i(n) = (1-\gamma)\alpha_i(n-1)+\gamma f_i(n-1) \tag{1}$$

where $0 < \gamma < 1$ is a smoothing constant (independent of the flow id i).

Like the regular TCP, DCTCP is not designed for any service differentiation and thus effectively results in equal division of the available bandwidth amongst the existing flows during the congestion episodes. As in regular TCP, such a division does not require knowing the actual available bandwidth. In particular, DCTCP reduces the window in proportion to the latest estimate of α_i such that in the limiting case of $\alpha_i = 1$, the window is halved. That is, the window control follows the following rule:

$$W_i(n) = W_i(n-1)\Big(1-\frac{\alpha_i}{2}\Big) \text{ if } \alpha_i > \varepsilon \tag{2}$$

$$W_i(n) = W_i(n-1)+1, \text{ if } \alpha_i \leq \varepsilon \tag{3}$$

where ε is a very small positive constant to address the fact that α_i will not become truly zero for a long time after the congestion has disappeared (i.e., after $f_i = 0$). This is not an issue in kernel implementations since all arithmetic is forced to be integer. The DCTCP RFC suggests marking threshold of $K > (RTT \times C)/7$, where C is the link capacity in packets per second.

The aggressive reduction of window based on congestion crossing a fixed threshold controls the latency well and the congestion proportional window cut achieves a better trade-off between the congestion and the utilization of the network, compared to other existing TCP variants like TCP Reno [9] or Westwood [10]. Note that α_i will generally be different for different flows, with a fatter flow experiencing higher α_i and thereby being subject to a greater reduction in throughput.

Some proposals have been made to use DCTCP in NVME-TCP protocol as a solution to prevent "incast collapse"[1]. Despite all the pros, DCTCP lacks providing differentiated treatment for different applications, which is very common in a data center environment. For example, in the presence of congestion, a high priority flow may suffer more compared

[1]https://www.snia.org/sites/default/files/ESF/SNIA-NVMe-TCP-Final.pdf

to other low priority flows if both of them react to the same congestion event similarly (halve the corresponding cwnd).

The D^{2}TCP proposal considers deadline of each class (or flow) while modulating the cwnd and thereby attempts to provide some level of service differentiation. In particular, for class i, it computes a ratio d_i of an estimated actual delay and the deadline, and skews the window modification by using $\alpha_i^{d_i}$ instead of α_i. In other respects, D2TCP works identical to DCTCP as shown in Table I. Note that the exponentiation required in D^{2}TCP is difficult to implement accurately in the kernel mode because of lack of floating point arithmetic. Also, a continuous monitoring of the remaining time to the deadline of an application level transfer is rather cumbersome to implement in the kernel.

Although D^{2}TCP provides differentiated services to some extent, it sufferes from some key limitations. First, the notion of "deadline" in D2TCP is rather simplistically defined; it is assumed that each flow will queue up all of the bytes it needs to transfer in one shot, and thus the deadline corresponds to the amount of time it takes to entirely transmit all of these bytes. Inherent in this view is the assumption that there are no overlaps between between successive "flows", i.e., all the bytes deposited are fully transmitted before another batch arrives. In reality, we need to handle applications that generate packets sporadically based on their incoming requests. The side effect of the peculiar deadline specification is that there is no notion of tail latencies, which is what we really would like to control in high speed remote storage access.

TABLE I
ALGORITHM COMPARISON OF D^{2}TCP AND DCTCP

D^{2}TCP	DCTCP
$\alpha_i = (1-\gamma)*\alpha_i+\gamma*F_i$ $\beta_i = \alpha_i^d$ where, γ= Smoothing Factor, F_i = ECN Fraction	$\alpha_i = (1-\gamma)*\alpha_i+\gamma*F_i$ $\beta_i = \alpha_i^d$ where, γ= Smoothing Factor, F_i = ECN Fraction
$d = T_c(n-1)/D$, where D = Deadline $T_c(n) = B/(\frac{3}{4}*W_i(n))$ where B = **Remaining bytes**	$d = 1$
$W_i(n) = W_i(n-1)[1-\frac{\beta_i}{2}]$, if $\beta_i > 0$ $W_i(n) = W_i(n-1)+1$, if $\beta_i = 0$	$W_i(n) = W_i(n-1)[1-\frac{\beta_i}{2}]$, if $\beta_i > 0$ $W_i(n) = W_i(n-1)+1$, if $\beta_i = 0$

III. QTCP – A QoS AWARE TCP WINDOW MODULUATION

We assume a fixed set of I QoS classes, with *a persistent TCP connection per class*. We assume that the applications are classified into one of these classes and thus each application uses a specific class throughout its lifetime, which is our notion of a "flow". (It is possible that long running applications have different phases with distinct behavior, and thus the same TCP connection could potentially require different treatment across phases, but we do not consider such situations.)

A. QoS Specification

In general, each class has a specified tail latency objective and a minimum bandwidth objective, e.g., at least 200 Mb/sec with 90 percentile latency of 100 μs. We consider two regimes of operation: 1. The bottleneck link has enough bandwidth to accomodate the average bandwidth demand of all the flows. Short-term congestion can occur in this case and can seriously affect the latencies achieved by various classes. Thus the objective of congestion control is entirely to satisfy the latency requirements that we assume are specified in terms of tail latencies.

2. The bottleneck link is lacking capacity to satisfy the minimum bandwidth of some of the flows. In this case, it is necessary to ensure that various clases get a predefined fraction of the bottleneck link capacity. We assume that the total available bandwidth of the bottleneck link is known here.

We further *assume that all tail latencies are specified using the same percentile value, e.g., 90 percentile*. If originally the latency is specified differently (e.g., 99 percentile), we then need some way to estimate the corresponding 90 percentile value. For example, if the mean and variance of the latency distribution is known, we can use Chebychev inequality to estimate the tail latency. That is, for a random variable X with given expected value $E[X]$ and standard deviation σ_X, we have:

$$Pr[|X-E[X]| \geq \delta\sigma_X] \leq 1/\delta^2 \text{ for any } \delta > 1 \qquad (4)$$

The reason for assuming the same percentile is that it enables us to control the window size based directly on the ratios of achieved and target latency values.

The BW based control is a little more complex. The problem is that the "desired bandwidths" must be limited if the total offered traffi exceeds the bottleneck link capacity C. Thus, there are two situations to consider for each class i:

1) No congestion: Desired BW = Offered load of the class i
2) Congestion: Desired BW = $C*$ Desired BW ratio for class i

B. Quality Factor and Window Flow Control

We now define a measure called *quality factor*, and denoted as Q_i for class i. Let L_{ia} and L_{it} denote, respectively, the actual and target tail latencies for class i. We express Q_i as a ratio of the two, with actual latency smoothed over time. That is,

$$Q_i = L_{it}/L'_{ia} \text{ where } L'_{ia} = (1-\gamma)L_{ia}+\gamma L'_{ia}, i = 1..K \quad (5)$$

where γ is the smoothing factor. Q_i is a dimensionless number, ranging from 0 to some maximum value limited by the admission control. If $Q_i > 1$, class i has a slack (i.e., its window can be squeezed), and if $Q_i < 1$, then class i has deficit, and its window needs to be increased. Since we assume that each class uses a separate TCP connection, the window for each class is controlled independently based on its Q factor.

A similar Q_i can be defined for bandwidth centric control, i.e.,

$$Q_i = \lambda'_{ia}/\lambda_{it} \text{ where } \lambda'_{ia} = (1-\gamma)\lambda_{ia}+\gamma\lambda'_{ia}, i = 1..K \quad (6)$$

where we have reversed the ratio, to keep the same sense for Q_i factor ($Q_i > 1$ means that we have slack, and $Q_i < 1$ means that we have deficit).

In addition to the quality factor, we also need to make use of the congestion feedback returned by the standard ECN mechanism.

An explicit latency based window control needs to decide how to measure the latency, how to convey it to the transmitter (since the latency is only known on the receive side), and how to use it for window control. There is also the question of how such a control will play with the ECN mechanism. The latency should include the transmit side queuing latency and TCP level network latency (send side processing from TCP down, transit delay through intermediate switches, propagation delay, and receive side processing up to TCP). Of these, the network latency will be identical for all classes unless the switches have the ability to do class specific markings.

The window modulation for different QTCP flows is done based on both the value of α_i (probability of congestion a.k.a. percentage of packets are ECN marked) and quality factor metric Q_i. The overall window modulation mechanism for a single flow i:

$$W_i(n) = \begin{cases} W_i(n)+1, & \text{No ECN} \\ W_i(n)(1-\frac{\alpha_i}{2}), & \alpha_i \geq 0 \text{ and } Q_i > 1 \\ W_i(n)(1-\frac{\alpha_i}{2})Q_i, & \alpha_i \geq 0 \text{ and } Q_i \leq 1 \end{cases} \quad (7)$$

The above scheme works as follows. When there is no congestion indication, then congestion window for each flows would increase by 1 per RTT. In case there is a congestion indication (marked ECN ACKs) in the previous RTT, then the window will be updated in proportion to α and Q_i. As discussed earlier, $Q_i > 1$ means, the flows has not satisfied the QOS requirement yet, so the modulation would only be based on the α. When $Q_i \leq 1$, the flow has already met the QoS demand, so it is okay to back off from the assigned bandwidth resources to make room for others.

C. QTCP in Presence of Regular TCP

The key aspect of QTCP is an attempt among flows belonging to different classes to adjust their windows according to the stated object (throughput ratio or latency). The data center network QoS is best supported in an environment where the bottlenecks can be monitored directly and conveyed to the relevant applications. A SDN or SDN-like infrastructure can indeed monitor and provide crucial information to applications such as the available bandwidth of the bottleneck link, location of the bottleneck link, the flows passing through any given switch port, etc. However, making such knowledge prerequisite to network QoS management is too restrictive. Therefore, we would like to minimize the nonlocal information that any class should know. The ideal situation is where each application determines independently what it needs to know to achieve its QoS target.

As discussed earlier, currently QTCP only needs to know the bottleneck bandwidth in order to decide the share of bandwdith that each class should get, but the location of the bottleneck link (or links) is not required. We henceforth assume that this quantity can be determined using the knowledge of network topology, the workload, and the data gathered by standard tools such as SNMP and/or explicit latency measurements by sending ICMP (ping) and traceroute messages. Such estimation is likely to be carried out at a rather low rate (e.g., periodically or when starting QTCP applications). In general, the QTCP flows of interest could be a subset of all flows, and thus the bottleneck bandwidth of interest is the one that includes other "uncontrolled" flows. Even more generally, one may have a mixed environment where some flows are QTCP while others follow an undifferentiated version of TCP such as DCTCP. In these cases, the overall bandwidth available to QTCP is the remaining bandwidth and the bottleneck from QTCP's perspective could well occur on a link that has a high overall bandwidth except that a significant portion of it is taken by non-QTCP flows.

The above low-rate estimation of available bandwidth may be inadequate in a dynamic environment where new non-QTCP applications may start or stop at any time. Therefore, we also consider an alternate mechanism where the QTCP itself continuously adjusts the bottleneck bandwidth by monitoring the impact of any interfering traffic. For this, we assume that the bottleneck bandwidth (λ) is known initially (given or estimated by other means). Then if an interfering flow alters this value, each class in QTCP estimates it as shown in Fig. 1. Here target_i is the QoS requirement of class i when there is congestion in the network, and actual_i is the estimated throughput till that point. The Q_i is the quality factor. Since Q_i quickly converges to close to 1 (as shown later), its perturbation by more than 5% is considered as a signal of a new interference or disappearance of the interference. We then change the target bandwidth by a the amount called "factor" in Fig. 1 and also adjust the overall bottleneck bandwidth λ. We then recompute the target bandwidth for each class using the given ratios. As we shall see later, the mechanism works quite well.

$W_i = W_i(1-\alpha/2)$
if $(Q_i > 1.05)$ // means there is interference flow
    factor = 0.95;
else if $(Q_i < 0.95)$ //interference flow left
    factor = 1.05;
else factor = 1.0
$\lambda = \lambda - (1-\mathrm{factor}) \times \mathrm{target}_i$
$\mathrm{target}_i = \mathrm{ratio}_i \times \lambda$; $Q_i = \mathrm{target}_i / \mathrm{actual}_i$
if $(Q_i < 1)$ $W_i = W_i Q_i$

Fig. 1. Estimation of BW Impact of Interfering Flows

Note that TCP-BBR [11] can estimate the bottleneck bandwidth, by observing the successive RTT variation and then by finding a optimal point of operation. Instead of limiting the data rate, BBR inject more packets than the Bandwidth Delay Product in order to find that optimal point. This allows full queue build up, which might cause significant performance degradation to other latency sensitive applications [12]. Our scheme works, because when any new TCP flow enters, due to the slow start phrase, changes in throughput (δ) occurs in a very slow rate. When this δ changes occurs, individual QTCP flow's can sense it by the fluctuations in quality factor. Our proposed scheme should work better if the δ can be estimated,

but this is beyond the scope of this paper.

IV. Analytic Modeling of QTCP

A. A Simple Operational Model

As with other versions of TCP, the essential aspects of QTCP behavior can be captured via a *discrete time model* (DTM) that considers the change in TCP window size from one round-trip time (RTT) to the next. The behavior may also be approximated by via *fluid flow model* (FFM) as in the analysis of DCTCP in [13]. Note that unlike DCTCP, where only one flow can be analyzed in isolation, the QTCP model must analyze a coupled system of equations involving all classes. Both models have their strengths and weaknesses. The main issue with DTM is that it considers the behavior of TCP only at certain discrete points and thus cannot model the intra-RTT state or window modulation changes. The main issue with FFM is that it incorrectly assumes infinitesimal control of state and because of that cannot easily handle the notion of state during the last RTT. In particular, the FFM in [13] uses an average value of RTT to refer to the last RTT cycle. We focus on DTM only in this paper.

For the DTM, we continue to use n as the current "time-slot" or RTT duration. Consider $i = 1..I$ active TCP connections, each belonging to a distinct class. Let C denote the capacity of the bottleneck link used by these classes with $C_i(n)$ as the share of class i at slot n. Obviously, with $\sum_{i=1}^{I} C_i = C$ in all cases. Let $R_i(n)$ the round-trip time (RTT), and $q_{aj}^{(i)}(n)$ the queue length of class j at the *bottleneck egress port of the switch as seen by an arriving class i packet*. Furthermore, let $p_i(n)$ denote the event that the switch queue is already at or beyond the threshold K when a class i packet arrives, and thus this packet has its CE bit set. Note that in the current window, the relevant event is from the last window. That is,

$$e_i(n) = \mathbb{I}_{\sum_{j=1}^{I} q_{aj}^{(i)}(n-1) \geq K} \tag{8}$$

In general, each arriving class i may see a different distribution of packets in the queue; however, since we assume that the switch uniformly marks packet of any class that sees a "full" queue, the dependence of $q_{aj}^{(i)}(n)$ is likely to be very weak, if any. Therefore, we henceforth assume that such a dependence does not exist, and denote the queue full event as simply $e(n)$. However, for a discrete time model, we need not the individual events but rather the probability of the queue being full, henceforth denoted as $p(n)$. We can estimate this as follows:

$$p(n) = \begin{cases} 1 - \frac{K-1}{B(n-1)} & \text{if } B(n-1) \geq K \\ 0 & \text{if } B(n-1) < K \end{cases} \tag{9}$$

where $B(n-1) = \sum_{i=1}^{I} q_{ai}(n-1)$).

The overall latency $L_{ai}(n)$ observed by an arriving class i packet is given by $L_{ai}(n) = d_i + q_{ai}(n)/C_i(n)$ where d_i is the baseline delay independent of queuing (including send/receive processing delays, link propagation delays, switch processing delays, and transmission time of one (arriving) request). We assume that the ACKs do not face any significant queuing delays, and thus $R_i(n) = d_i'$ where d_i' is the backwards delay. For simplicity we will assume that $d_i' = d_i$ for all i.

We assume that a suitable admission control is in place so that the *average* total offered traffic to the bottleneck link is always strictly less than the the link bandwidth C. In other words, we assume that the packets cannot build up at the transmit nodes indefinitely. Thus, the congestion is a result of the burstiness in individual class traffic, including the overlaps in high traffic periods of various classes such that the link capacity is temporarily exceeded but no packet is ever dropped either in the switches or at the transmitter. That is, the long term throughput of the system equals the offered load.

Throughout Ratios: Class i is targeted to get the given BW ratio of r_i relative to class 1 (i.e., $r_1 = 1$). That is, $C_i = C.r_i / \sum_i r_i$ and is independent of slot. Since no packets are lost, the actual throughput can be estimated from the number of packets transmitted in the last window, i.e., $\lambda_i(n) = W_i(t - \overline{R_i})/R_i(t - \overline{R_i})$. Therefore, $Q_i(n) = \lambda_i(n)/C_i$.

Queuing Latency: We assume that the classes are ordered according to an importance score, with class 1 being the most important. Then we require that the latency requirement applies only to classes $i < I' < I$ where I' is such that classes $i \in [1..I']$ do not occupy more than 50% of the link BW C. The queuing latency target L_{it} for these classes must be chosen sufficiently large to be realizable, particularly since the switch is assumed to use FCFS scheduling for all packets.[2] For classes $i \in [I'+1..I]$, the target latency may be left unspecified must be set to a large value representing the worst possible congestion. The actual congestion $L_i(n) = d + q_{ai}(t - \overline{R_i})/C_i(t - \overline{R_i})$ where $C_i(n) = W_i(n)/R_i(n)$. Therefore, $Q_i(n) = L_i(n)/L_{it}$.

We could then write the equations for all quantities. In the following we assume that the bandwidth, window size and throughput are in the units of packets rather than bytes. Based on the discussion above, we first restate the basic quantities below.

$$C_i(n) = \begin{cases} C.r_i / \sum_i r_i & \text{Throughput control} \\ \frac{W_i(n-1)}{R_i(n-1)} & \text{Latency Control} \end{cases} \tag{10}$$

$$Q_i(n) = \begin{cases} \frac{C_i(n) R_i(n-1)}{W_i(n-1)} & \text{Throughput control} \\ \frac{d_i + q_{ai}(n-1)/C_i(n-1)}{L_{it}} & \text{Latency Control} \end{cases} \tag{11}$$

$$\alpha_i(n) = \alpha_i(n-1) + \gamma[p(n) - \alpha_i(n-1)] \tag{12}$$

The order of calculation is as follows: We first estimate $p(n)$, i.e., whether ECN was received in the last window, based on the queue length in the the previous slot ($q_{ai}(n-1)$). This, in turn, is used to compute the fraction of BW given to each flow in current slot, $C_i(n)$, and from there the quality factor $Q_i(n)$. We next update α, which in turn provides all the parameters required to update the window size $W_i(n)$ and the RTT ($R_i(n)$) for the current slot. This, in turn, is then used to estimate $q_{ai}(n)$, the the queue length for the current slot, so

[2] One could use multi-class open-system queuing formulae [14] to estimate range of values to use.

that the temporal evolution can continue.

$$W_i(n) = \begin{cases} W_i(n-1)+1 & \alpha_i(n) \leq \varepsilon \\ W_i(n-1)[1-\frac{\alpha_i(n)}{2}] & \alpha_i(n) > \varepsilon \& Q_i(n) > 1 \\ W_i(n-1)[1-\frac{\alpha_i(n)}{2}]Q_i(n) & \alpha_i(n) > \varepsilon \& Q_i(n) \leq 1 \end{cases} \quad (13)$$

$$R_i(n) = 2d_i + B(n-1)/C \quad (14)$$

$$q_{ai}(n) = max[0, q_{ai}(n-1) + W_i(n) - C_i(n-1)R_i(n)] \quad (15)$$

where we have used $C(n-1)$ in the last equation, since the known drainage rate is $C(n-1)$ during nth slot.

These equations can be solved recursively starting with some initial conditions. For example, we can assume that $q_{ai}(0) = K/I$ (all flow have equal number of packets at the switches) or $q_{ai}(0) = 1$ (only one packet in transit at the switch), and $C_i(0)$ are given as the desired throughput ratio or inversely proportional to the desired latency. We could then determine $R_i(0)$. Then, $W_i(0) = R_i(0)C/I$ for all $i \in 1..I$ (all flows have equal window size). We also assume $\alpha_i(0) = 0$, i.e., no congestion is encountered initially. Note that all classes must be handled together since we need the summation of queue lengths for computing $p(n)$ and $R_i(n)$.

One could seek the "steady state" from these equations by considering the case where the window and RTTs do not change from slot to slot. However, it is clear from equation (13) that this system does not have any fixed point.

The equations above assume that there are always enough packets available at the transmitter so that it can fill whatever the window size is in each slot. We can extend the model further by including a packet generation process and keeping track of untransmitted packets for each class i, henceforth denoted as $U_i(n)$. The actual window size for class i, henceforth denoted as $W_i'(n)$, is then the minimum of the computed window size $W_i(n)$ (from $W'(n-1)$ using equation like (13)). That is,

$$M = \inf_{(\sum_{m=1}^{M'} G_i^{(m)}) > R(n-1)} (M') \quad (16)$$

$$U_i(n) = U_i(n-1) - W_i'(n-1) + M - 1 \quad (17)$$

$$W'(n) = \min[W(n), U_i(n)] \quad (18)$$

where $G_i^{(m)}$ denotes the time between mth and $(m-1)$st packet during an RTT. This gap is obviously driven by the packet arrival process which could be bursty.

B. Comparison of Model and Simulation

We validate the analytical modeling of QTCP with the ns3 simulation in Fig. 2 with 3 applications. We set the bottleneck bandwidth C at 10 Gbps; the applications are injecting traffic at a rate of 3, 6 and 9 Gbps respectively. We assume the threshold K to be 140 ($K \sim 0.17Cd$, where C = Bottleneck capacity, d= propagation delay), which is also used in the DCTCP paper [7].

From Fig. 2 we can observe that the our analytical model shows the similar behavior in terms of throughput of the individual applications, as compared to the ns3 simulations. Thus

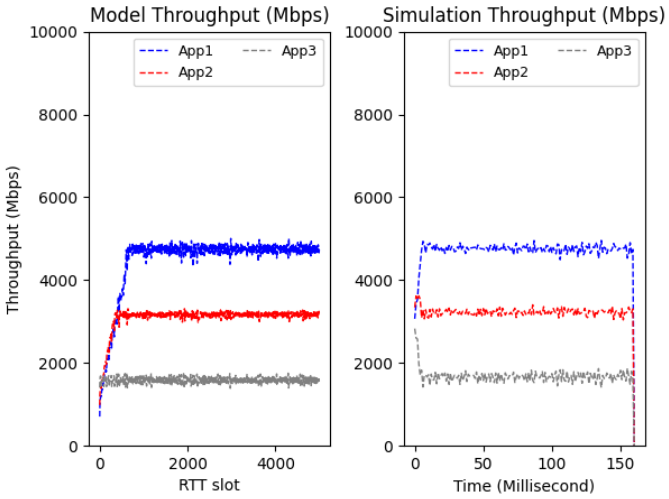

Fig. 2. Comparison of throughput between model and simulation

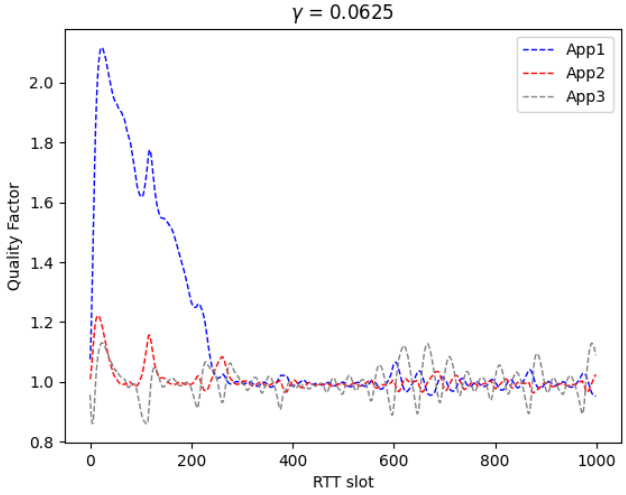

Fig. 3. Variation of quality factor per RTT

validates that that our analytical model closely approximates the behaviour that is obtained from the simulations.

C. Convergence and stability analysis

We next conduct the convergence and stability analysis of the developed analytical model in presence of 3 applications. Fig. 3 shows the variation of Q_i with RTT slots, where the RTT and γ are assumed to be 248μs and 0.0625 respectively. From this figure we can observe that the Q_i's of all three applications converge to approximately 1 within 100 RTT slots. Because of the target throughput based window modulation, the actual throughput of the applications reaches close to the target throughput, which makes the quality factors close to unity.

Fig. 4 shows the effect of different RTT values on the convergence time, where γ is assumed to be 0.0625. As expected the convergence time increases with the increase in RTT values. In a data center environment, the RTT is relatively

small ($\sim 150-200$ μs) [15]; thus, the convergence time will be fairly quick. Fig. 4 also demonstrates that the quality factor of the applications will converge to 1.

The variation of the quality factor with different smoothing constants γ is depicted in Fig. 5 with RTT kept at 220 μs. We can observe that increasing γ results in quicker convergence due to aggressive window management. We don't set γ value too high (~ 1.0), as under burst traffic situation there will be huge fluctuation in the congestion window (variance in both RTT and quality factor), which would cause response jitters and as a result performance degradation.

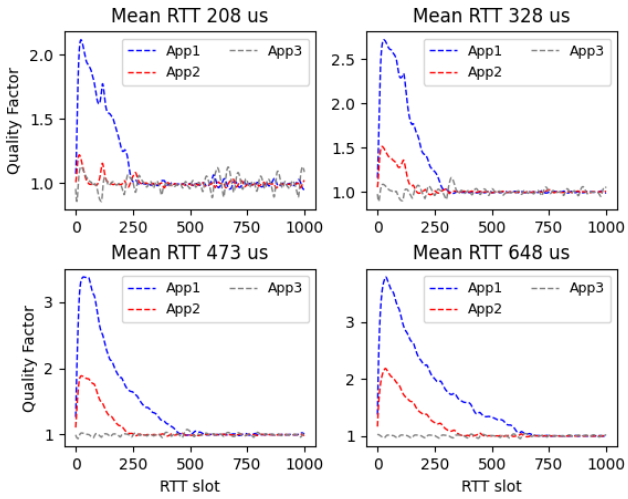

Fig. 4. QTCP convergence with different RTT

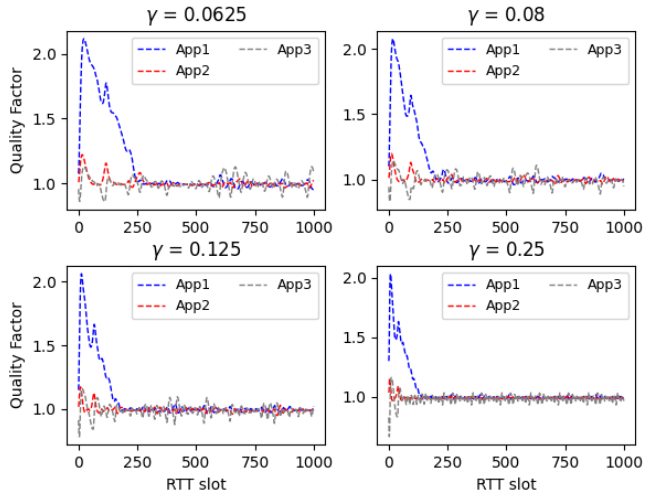

Fig. 5. QTCP convergence with different γ

V. Performance Evaluation

We comprehensively evaluate the QTCP mechanism using the popular ns3 network simulation package. For this, we started with the detailed DCTCP implementation in ns3 (which closely follows the RFC 8259) and implemented the proposed quality aware window modulation mechanism. Our simulation environment consists of 4 servers and 2 switches as shown in

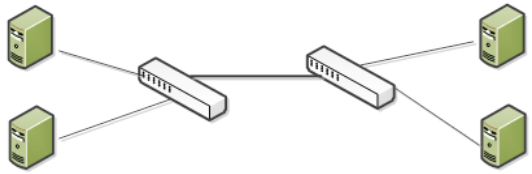

Fig. 6. Simulation setup; all links are connected through 10 Gbps port

Fig. 6. All the links are of 10 Gbps. These switches have the ECN capability, threshold K set as 140.

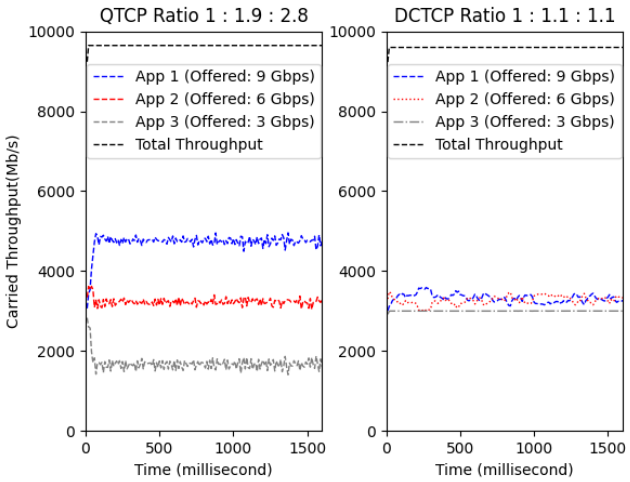

Fig. 7. Comparison of bandwidth sharing between QTCP and DCTCP

A. Comparison with DCTCP Throughput

Fig. 7 shows the comparison between QTCP and DCTCP in presence of 3 applications carrying load of 3, 6, and 9 Gbps respectively; thus the accumulated offered load (i.e. 18 Gbps) exceeds the bottleneck link capacity of 10 Gbps. The ratio of offered load is 1: 2: 3. In case of DCTCP, we can observe equal sharing of the bandwidth between 3 applications in Fig. 7(b), and the carried throughput ratio is 1 : 1.1 :1.1. In case of QTCP, the carried throughput ratio is almost exactly equal to that of the target ratio. Thus QTCP can achieve the desired QoS objective of allocating the bottleneck link bandwidth in the desired proportions to different classes. Also notice that, the overall throughput in case of QTCP is almost equal to DCTCP (9.63 Gbps compared to 9.57 Gbps in DCTCP).

B. RTT fairness Comparison

We define RTT fairness as the extent to which we can achieve the target throughput ratios, when the RTT of different applications flows are different. Fig. 8 shows the result of bandwidth sharing between classes with different RTTs in case of DCTCP. The average RTT of high QoS Application (9 Gbps) is approximately 1800 μs, whereas the low QoS applications (i.e. 3 Gbps and 6 Gbps) have a RTT of 350 μs [3]. Although the conventional goal of RTT fairness is to equalize

[3]We simulate different RTTs by changing the number of hops in ns3

the bandwidth sharing when different flows experience different RTT, DCTCP shows bias against flows with longer RTT, as flows with shorter RTTs grab bandwidth more quickly.

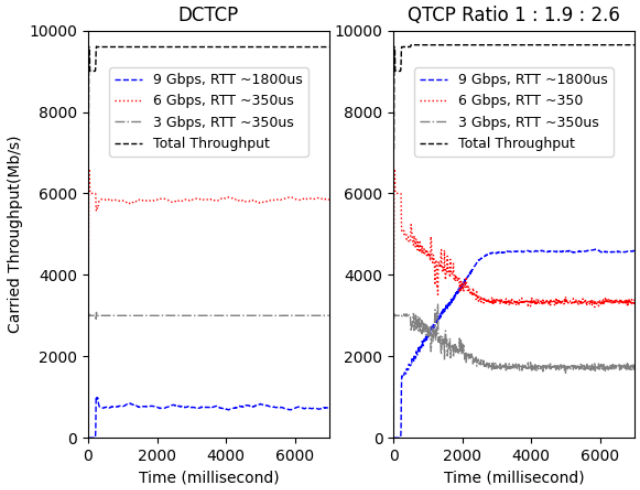

Fig. 8. RTT Fairness in DCTCP and QTCP

Fig. 8 also shows the result of QTCP using the same topology set up and the workload. The bandwidth distribution ratio is close to the target throughput ratio in case of QTCP, though it requires some time to stabilize. Also notice that, QTCP offers the same overall throughput as DCTCP (9.6 Gbps). In QTCP, although the flows with shorter RTTs grab window more quickly, the quality factor insists the low QoS flows to make room for the high QoS flows, regardless of the RTT.

C. DCTCP friendliness

Since not all flows in a data center may use QTCP, it is important to study what happens if QTCP competes against DCTCP. For this, we introduce one DCTCP flow that coexists with our QTCP flows and verify whether the committed ratio amongst QTCP flows persists. Fig. 9(a) shows that the QTCP flows maintain their respective ratios even in the presence of DCTCP flows. When the DCTCP flow leaves (after about 200 of millisecond of simulation time), the QTCP flows manage to grab the bandwidth resources according to the QoS specified ratio (1: 2: 3). In Fig. 9(b) we simulate the scenario where the DCTCP flow enters and leaves during the simulation; in this scenario also we observe that QTCP adjusts the remaining bandwidth among the active flows. Another solution to ensure the DCTCP friendliness could be to reserve the bandwidth explicitly for the interfering flows as suggested in [16]. However, this will result in network under-utilization when there is no interfering DCTCP flow.

D. Latency Comparison with DCTCP and D^{2}TCP

We now consider the case of latency sensitive traffic, where the QoS is defined in terms of target latency. For the experiments, we categorized applications into four classes; three of four classes have latency requirements of 5366, 6604, 7832

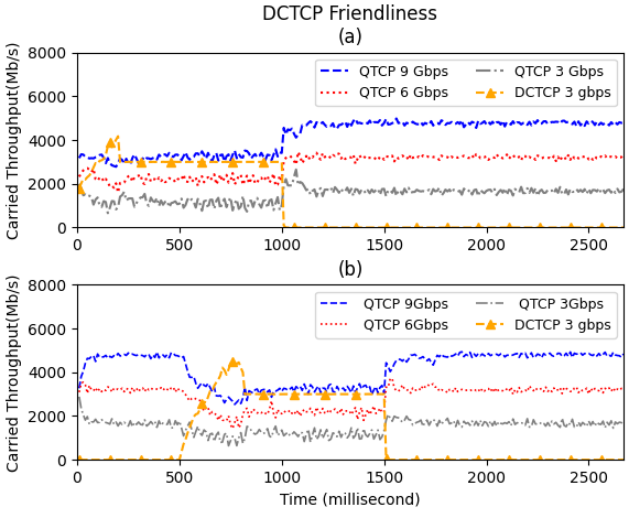

Fig. 9. Illustration of DCTCP friendliness of QTCP

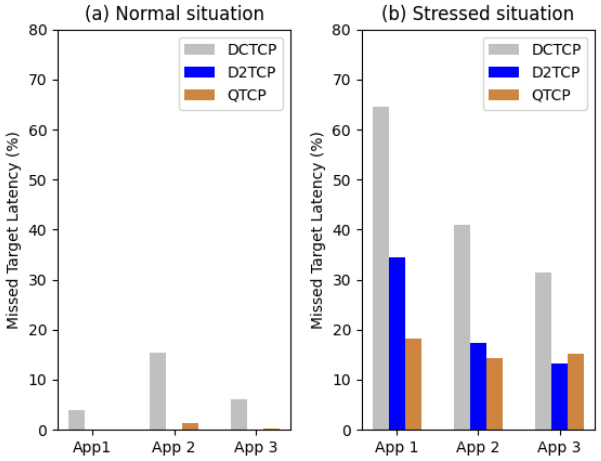

Fig. 10. Comparison of QTCP, DCTCP and D^{2}TCP

μs respectively, whereas class four has no QoS requirements. The mean transfer size we choose is 2MB.

Fig. 10 shows the comparison between QTCP, DCTCP and D^{2}TCP for latency sensitive applications. We simulate both the normal and stress situation to show the effectiveness of our scheme. Fig. 10(a) depicts the normal situation where the high priority flows (i.e. applications 1 and 2) generate packets at a frequency lower than that of others; in our simulations their overall generated traffic is 10% and 20% respectively. In case of QTCP, almost all the applications are able to meet their target latency, whereas in DCTCP ~5-8% packets miss their deadlines. In 10(b) we simulate a more challenging situation, where each class contributes to 25% of the overall load. As compared to DCTCP, in case of QTCP the fraction of traffic with missed target latency reduces from ~30-65% to ~15-18%. Notice that in Fig. 10, we do not report the latency statistics for application 4, as it is assumed to be the assured class with no QoS requirement.

Fig. 10 also compares the performance of QTCP with D^{2}TCP. In the normal scenario, D^{2}TCP does not miss any

deadlines. However, in stress situation QTCP ensures lesser deadline misses. As compared to D^{2}TCP, QTCP reduces the percentage of missed deadlines from ∼35% to ∼18% for application 1, whereas for the other applications the performance of both the schemes are almost identical.

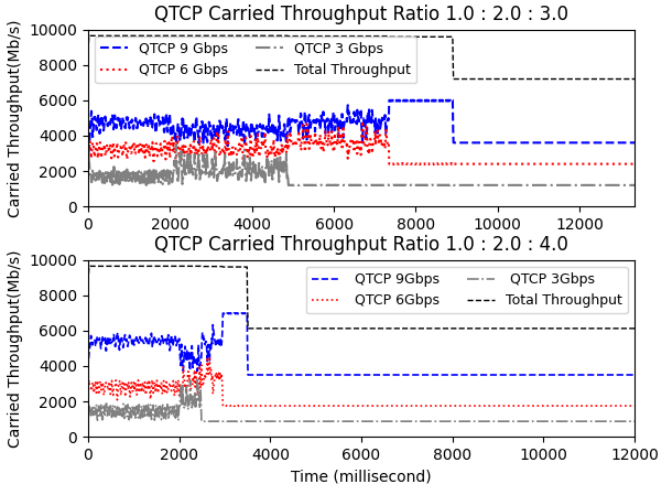

Fig. 11. Illustration of the adaptive nature of QTCP; in both scenarios the system transitions from congestion scenario to congestion free scenario.

Fig. 11 confirms that QTCP can adapt to the network load, as expected. In Fig. 11(top) the offered load is initially 3, 6 and 9 Gbps respectively for the three classes. After a while, all offered loads are halved (i.e., 1.5, 3.0 and 4.5 Gbps respectively), so that the overall load becomes less than the link capacity. The figure shows how QTCP adapts smoothly from high congestion scenario to congestion-free scenario, while maintaining the throughput ratios approximately 1:2:3 among the applications. Fig. 11(down) shows the similar behaviour with initial loads of 1.75, 3.5 and 7 Gbps respectively.

E. Impact of Q_i update interval

In the QTCP analytic model, we assumed that the Q_i is updated in every RTT. But such a fine grained control affects the stability of different flows. In Fig. 12 we show the impact of different update intervals on the QTCP carried throughput via simulation. We use the same topology used to validate RTT Fairness. It is clear that the convergence time for different flows are almost similar for different intervals, approximately 2500 milliseconds. However, the fluctuations around the average value show an interesting behavior. When the update is done every RTT, the fluctuations are high, but almost disappear if updates are done every 10 RTT. Further increasing the update interval actually increases the fluctuations, but they are much more consistent than for the 1 RTT case. The reason for this behavior is that a large update interval results in delayed action which allows the window and hence the throughput to swing significantly before it is controlled. The eventual control in this case overcompensates thereby causing oscillations. Based on these observations, we have chosen an update interval of 10 RTTs for much of our experiments.

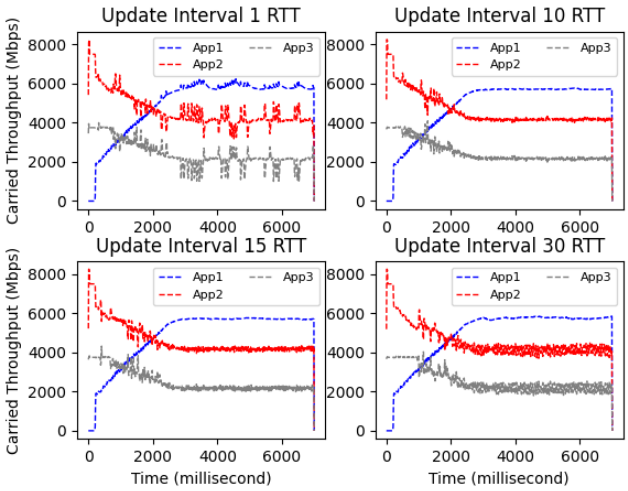

Fig. 12. Carried throughput over time for different Q_i update intervals

F. Class Based Bandwidth Distribution

We next consider a scenario where there are 3 classes of traffic, each of them consisting of 5 applications; each application in the same class is treated equally. Here the target throughput of these 3 classes are assumed to have a ratio of 1:2:3. Fig. 13(a) shows the carried throughput of the first application of each classes. Fig. 13(b) shows the collective carried throughput of the applications in these 3 classes, which also demonstrates that the throughput ratio follows the expected target ratios.

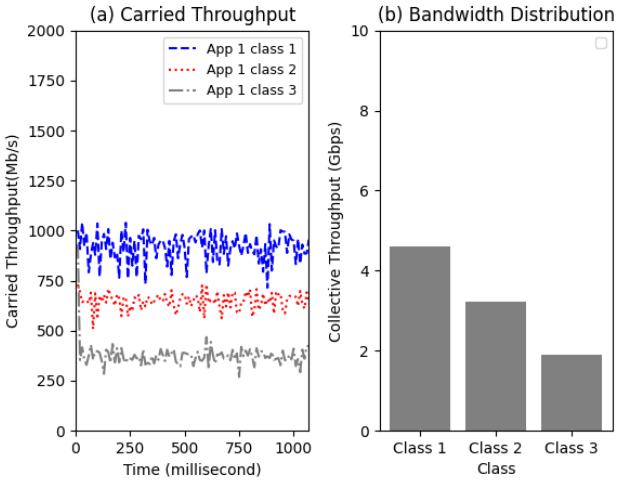

Fig. 13. Bandwidth distribution of 3 classes of traffic, each consisting of 5 applications

VI. Related Work

Although the main goal of conventional TCP is fairness (equal sharing of bottleneck bandwidth) amongst different flows, some variants address the differentiated treatment. Differentiated treatment in Transport layer can be classified into two categories: credit based control, and window modulation

based control. In this section, we summarize some of most relevant works in both categories.

Credit Based Control: Expresspass [17], Reflex [18] are credit based congestion control, which offer differentiated admission control by generating tokens in proportion to the target requirement. The transmitter, receiver, and switches coordinate to control the credit packets (tokens) per flow basis, which essentially determines the available bandwidth for data packets in the reverse direction. However, credit based solution requires changes in the protocol and specialized hardware to support token exchange operations. In [19]–[21] the authors have focused on QoS aware flow admission control; however, these studies are not in the TCP context.

Window Modulation Based Control: Reference [22] is based on the loss based TCP variant CUBIC [2], and ensures low latency for different cellular applications. Their goal is to emulate the AQM behavior at the server end. They estimate network status by observing the variance in RTT (due to packet drops), and modulate the window by comparing the target against the actual RTT. In our case the RTT variation is not significant due to the high bandwidth links, and no packets are actually dropped. So this scheme is not applicable to us. In the context of data center TCP environment, Homa [23], L^{2}DCT [24], D^{2}TCP [8], PDQ [25], D^3 [26] consider QoS in terms of individual flow completion time (i.e. deadline). Homa addresses head-of-the-line (HoL) blocking issue posed by TCP streams. They leverage in-network queue priority to provide low latency QoS to the small messages (99 percentile latency of 10 μs). L^{2}DCT and D^{2}TCP modulate TCP congestion window for different flows based on the QoS parameter provided. One of the key issue with these schemes is that the administrators need to have prior knowledge about the network delay and RTT in order to set the QoS parameters, whereas in case of QTCP we just need to specify the relative bandwidth ratio of different flows. PDQ proposes distributed scheduling algorithm, where the switches coordinate among themselves to schedule the high priority flow earlier (i.e. flow with critical deadline). It requires specialized switches and extensions to the protocol header to convey the QoS hints. D^3 is another deadline-aware TCP variant, however, D^3 [27] requires specialized switches and is not feasible for a ubiquitous solution. D^3 also requires centralized control, so scalability might get affected badly by the communication overhead.

VII. Conclusions and Future Work

In this paper we develop a lossless, QoS aware TCP for data center storage, that offers fairness by distributing the network bandwidth to different applications according to the relative QoS requirements. QTCP can be implemented in ECN-capable switches and does not need any specialized hardware. Using comprehensive simulations, we have shown that QTCP ensures RTT fairness, can co-exist with DCTCP, and supports better differentiated treatment among the applications as compared to DCTCP and D^{2}TCP mechanisms. In the future we would like to implement QTCP in the Linux kernel and test its efficacy with real servers and data center switches supporting ECN. We will also explore various combinations of throughput and latency related QoS requirements such as mixture of classes with both types of requirements or a latency requirement along with a minimum committed rate requirement.

References

[1] N. Parvez, A. Mahanti, and C. Williamson, "An analytic throughput model for tcp newreno," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 448–461, 2010.

[2] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," p. 64–74, 2008.

[3] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through intel dpdk," in *European Workshop on Software Defined Networks*, 2014, pp. 1–6.

[4] D. Wisniewski, "Solar flair: An open-road challenge," *IEEE Potentials*, vol. 29, no. 1, pp. 6–9, 2010.

[5] S. Zheng, M. Hoseinzadeh, and S. Swanson, "Ziggurat: A tiered file system for non-volatile main memories and disks," in *USENIX FAST*, 2019.

[6] M. Hagen, "Data center bridging tutorial," *University of New Hampshire—InterOperability Laboratory*, pp. 1–3, 2009.

[7] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010.

[8] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *ACM SIGCOMM*, 2012.

[9] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling tcp reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, pp. 133–145, 2000.

[10] M. Gerla *et al.*, "Tcp westwood: congestion window control using bandwidth estimation," in *IEEE GLOBECOM*, 2001, pp. 1698–1702.

[11] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *ACM Queue*, vol. 14, pp. 20–53, 2016.

[12] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to use and when not to use bbr: An empirical analysis and evaluation study," in *ACM IMC*, 2019.

[13] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of dctcp: Stability, convergence, and fairness," in *ACM SIGMETRICS*, 2011, pp. 73—-84.

[14] K. Kant, *Introduction to computer system performance evaluation*. McGraw-Hill, 1992.

[15] G. Zeng *et al.*, "Combining ecn and rtt for datacenter transport," in *APNet*, 2017.

[16] C. Guo *et al.*, "Rdma over commodity ethernet at scale," in *ACM SIGCOMM*, 2016.

[17] I. Cho, D. Han, and K. Jang, "Expresspass: End-to-end credit-based congestion control for datacenters," *ArXiv*, 2016.

[18] A. Klimovic, H. Litz, and C. Kozyrakis, "Reflex: Remote flash $\approx$ local flash," *SIGARCH Comput. Archit. News*, p. 345–359, 2017.

[19] T. Zhu, D. S. Berger, and M. Harchol-Balter, "Snc-meister: Admitting more tenants with tail latency slos," in *SoCC*, 2016.

[20] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Prioritymeister: Tail latency qos for shared networked storage," in *SoCC*, 2014.

[21] E. Thereska *et al.*, "Ioflow: A software-defined storage architecture," in *ACM SOSP*, 2013.

[22] S. Abbasloo, Y. Xu, and H. J. Chao, "C2tcp: A flexible cellular tcp to meet stringent delay requirements," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 4, pp. 918–932, 2019.

[23] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *ACM SIGCOMM*, 2018.

[24] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *IEEE INFOCOM*, 2013.

[25] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *ACM SIGCOMM*, 2012.

[26] C. Wilson, H. Ballani, T. Karagiannis, and A. I. T. Rowstron, "Better never than late: meeting deadlines in datacenter networks," in *ACM SIGCOMM*, 2011, pp. 50–61.

[27] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM*, 2011.