Provisioning Differentiated QoS for NVMe over Fabrics

Joyanta Biswas*, Jit Gupta*, Krishna Kant*, Amitangshu Pal†, Dave Minturn‡
*Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
†Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, India
‡Intel Corp, Hillsboro, OR 97291, USA

***Abstract*—In this paper, we propose a quality of service (QoS) aware transport solution for storage access over data center network. Motivation for QoS differentiation comes from the emerging storage technologies which not only provide network comparable latency but also overwhelm the network bandwidth close to the storage servers. We consider both throughput and latency related QoS requirements and demonstrate how they can be enforced by Explicit Congestion Notification (ECN) enabled switches. The mechanism can be viewed as a way of adding QoS capability to the existing data center transport solutions. Our scheme can co-exist with existing congestion management schemes and ensures RTT fairness while providing service differentiation. We show that our scheme can provide differentiated treatment and besides achieving better throughput fairness during the congestion episode, our solution can reduce the target latency misses by up to 71% and 80% as compared to the existing TCP and RDMA transport.**

I. Introduction and Motivation

Traditionally, storage devices and the storage access protocols have been rather slow and thus the network latency in remote access storage has not been an issue. However, with the emergence of high performance storage devices and emerging remote access protocols such as NVMe over Fabrics (NVMe-oF) [1], network congestion is becoming a significant issue [2]–[4].[1] For example, even a cheap consumer SSD can drive ∼25-35 Gb/sec of throughput. The Mellanox NVMe-oF performance report shows that 4 NVMe SSDs can saturate 100 Gb/sec links in time, whereas 250 SATA HDDs are required to saturate the same links [5]. Thus, the end-to-end quality of service (QoS) is becoming essential for storage access, and the network becomes a crucial piece of it.

NVMe-oF establishes the host to target connection and thus needs an underlying transport layer such as TCP or RDMA (remote direct memory access). Data centers currently use both Layer2 (L2) and Layer3 (L3) switches; therefore, an end-to-end QoS is best implemented at the transport level. This may, in turn, use features provided by lower layers, if available, but we focus on the transport layer only in this paper. Two examples of lower level QoS features are (a) the data center bridging (DCB) [6], which includes the priority flow control (PFC) and enhanced transmission selection (ETS), and (b) IP level DSCP (differentiated services code point) [7]. While DCB is becoming more generally available in data center switches, it must be augmented with additional mechanisms to provide an end-to-end solution. The DSCP mechanism is not very useful because (a) it is defined in terms of packet losses, which are highly undesirable in data centers, and (b) it is a hop-by-hop control, typically implemented in the routers [8].

For NVMe-oF transport, we shall consider both TCP and RDMA transports. TCP is predominant in data centers; however, because of its kernel-based and software-centric implementation, it results in high latencies. RDMA, in contrast, is largely supported by the hardware, is much leaner, and also avoids kernel transitions. RDMA is essential for access to remote persistent memory (PM) and to other emerging high-speed technologies where low latency is crucial [9].

Because of the undesirability of packet losses, we shall focus on lossless versions of TCP where the congestion control is initiated before any loss can occur. A popular and widely implemented version in this regard is the Data Center TCP (DCTCP) [10], which relies on the ECN (explicit congestion notification) mechanism for congestion feedback, but in a lossless way. In particular, if the switch buffer occupancy exceeds some threshold much below the actual buffer size, the congestion is indicated, and the source then aggressively reduces the flow so that the congestion is contained quickly and decisively. The ECN mechanism involves two bits; the Congestion Encountered (CE) bit in a packet is set when it encounters congestion on some switch in the path. When the packet reaches the receiver, and the CE bit is set, the receiver sets the ECN-echo (ECE) bit in a reverse packet (such as ACK) to inform the source of the congestion. However, DCTCP does not provide differentiated service during congestion. Thus a goal of this paper is to propose a DCTCP-like mechanism called *Quality Aware TCP (QTCP)* that allows differentiated treatment to TCP connections belonging to different QoS classes that happen to pass through a bottleneck link.

On the other hand, RDMA was originally developed for the InfiniBand (IB) technology and implemented entirely in hardware for low-latency lossless transfers. It has been successfully transported to Ethernet. In particular, RDMA over Converged Ethernet (RoCEv1) is an Ethernet layer protocol that allows IB traffic to be Ethernet (L2) compatible. Its successor, RoCEv2 [11], replaces the IB header with the UDP and IPv4/IPv6 header, eliminating the need for the IB stack entirely. It also allows RDMA to utilize the feature of both L2 (Ethernet) and L3 (IP). Data Center Quantized Congestion Notification (DCQCN) is an end-to-end congestion control protocol that works with ROCEv2, and leverages both the PFC and ECN for congestion management. Like DCTCP, DCQCN also lacks differentiated service during congestion. Our work addresses this limitation and introduces a *Quality*

[1]NVMe is a hardware-supported, low-latency storage access protocol that is becoming ubiquitous, and NVMe-oF is its extension that essentially handles NVMe requests over the network.

Aware RDMA (QRDMA) mechanism similar to QTCP, but in the context of RDMA.

The novelty of this paper is to develop QTCP and QRDMA *specifically targeted for handling NVMe-oF related storage traffic flows inside the confines of the data center.* Although it is possible to use them elsewhere in the data center as well, that aspect is beyond the scope of this paper. Therefore, we do examine the coexistence of QTCP/QRDMA with flows using the regular (non-differentiated) versions of these protocols.

The rest of the paper is organized as follows. We first provide an in-depth discussion of how QoS can be infused in DCTCP, its analytic modeling, and the corresponding results. In particular, section II discusses the limitations of DCTCP and introduces QTCP. Analytical modeling of QTCP is explored in section III and the evaluation results are covered in Section IV. Section V then discusses QRDMA briefly and shows the results. Because of the significant similarities in the design of QTCP and QRDMA, we keep the discussion of the latter short. Section VI summarizes the related works. We conclude the paper in Section VII.

II. QoS Differentiation in QTCP

A. Overview and Limitations of DCTCP

DCTCP uses 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 acknowledgments in a window that arrive with the ECE bit set. Consider I competing TCP connections (or flows). Let $f_i(n)$ denotes 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:

$$\alpha_i(n) = (1-g)\alpha_i(n-1) + gf_i(n-1) \quad (1)$$

where $0 < g < 1$ is a smoothing constant (independent of the flow id i). DCTCP reduces the window per round-trip time (RTT) 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:

$$W_i(n) = W_i(n-1)\Big(1 - \frac{\alpha_i}{2}\Big) \quad (2)$$

Despite all the pros, DCTCP is not designed for any service differentiation and thus effectively results in an equal division of the available bandwidth amongst the existing flows during the congestion episodes. As a result, in the presence of congestion, a high priority flow may suffer more as compared to other low priority flows if both of them react to the same congestion event similarly. That means, regardless of the QoS management in NVMe storage protocol (e.g. queue arbitration, block layer flow control [1]), the end-to-end QoS objective remains unattainable.

B. QoS Specification

We assume a fixed set of I QoS classes and all the existing applications are classified into one of these classes. Thus each application uses a specific class throughout in its lifetime, which is our notion of a "flow". In general, each class has a specified tail latency objective and a minimum bandwidth objective, e.g., at least 200 Mb/sec with a 90 percentile latency of 100 μs. We consider two regimes of operation:

1. The bottleneck link has enough bandwidth to accommodate the average bandwidth demand of all the flows.

2. The bottleneck link is lacking the capacity to satisfy the minimum bandwidth of some of the flows. In this case, it is necessary to ensure that various classes get a predefined fraction of the bottleneck link capacity. We assume that the total available bandwidth of the bottleneck link is known here, which can be estimated from the techniques used in [12].

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 are 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:

$$Pr[|X - E[X]| \geq \delta\sigma_X] \leq 1/\delta^2 \text{ for any } \delta > 1 \quad (3)$$

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

The "desired bandwidths" of a flow must be limited if the total offered traffic 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 = $C \times$ Desired BW ratio for class i

The congestion state and extent of congestion can be estimated using the ECN mechanism explained earlier.

C. Quality Factor and Flow Rate Control of QTCP

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

$$Q_i = L_{it}/L'_{ia} \text{ where } L'_{ia} = (1-\gamma)L_{ia} + \gamma L'_{ia}, i = 1..K \quad (4)$$

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 $Q_i > 1$, corresponding flow has a slack (i.e., its window can be squeezed), and if $Q_i < 1$, then flow has a deficit, and its window needs to be increased. Since we assume that each flow uses a separate connection, the flow rate for each of them is controlled independently based on its Q_i.

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

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

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

The flow rate modulation for different QoS aware flows is done based on both the value of α_i (probability of congestion a.k.a. percentage of packets that are ECN marked) and their

Q_i. If the current window size of a flow is W_i, then the overall flow rate control mechanism for a single flow is:

$$W_i(n) = \begin{cases} W_i(n)+1, & \text{No ECN} \\ W_i(n)(1-\frac{\alpha_i}{2}), & \alpha_i \geq 0 \text{ and } Q_i > 1 \\ W_i(n)(1-\frac{\alpha_i}{2})Q_i, & \alpha_i \geq 0 \text{ and } Q_i \leq 1 \end{cases} \quad (6)$$

The above scheme works in every update interval (i.e. RTT) as follows. When there is no congestion indication, the window size of each flow would increase by 1 unit. In case there is a congestion indication (marked ECN packets from the receiver) during successive update intervals, $W_i(n)$ will be updated in proportion to α_i and Q_i. As discussed earlier, $Q_i > 1$ means that the flows have not satisfied the QoS requirement yet, so the modulation would only be based on the α_i. 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.

III. Analytical Modeling of QTCP

A. A Simple Operational Model

The essential aspects of QTCP can be captured via a *discrete time model* (DTM). Our proposed DTM model considers the change in flow rate from one update interval to the next. The behavior may also be approximated via a *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, our 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-update state. On the other hand, 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 update. 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 current update duration. Consider $i = 1..I$ active connections, each belonging to a distinct class, so the notion of class and flow can be used interchangeably. Let C denotes the capacity of the bottleneck link used by these classes with $C_i(n)$ as the share of class i at slot n, with $\sum_{i=1}^{I} C_i = C$. Let $R_i(n)$ denote the round-trip time (RTT), and $q_{aj}^{(i)}(n)$ denote 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,

$$e_i(n) = \mathbb{I}_{\sum_{j=1}^{I} q_{aj}^{(i)}(n-1) \geq K} \quad (7)$$

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 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 DTM, we need not observe the individual events but rather the probability of the queue being full, henceforth denoted as $p(n)$. We can estimate this as follows:

$$p(n) = \begin{cases} 1 - \frac{K-1}{B(n-1)} & \text{if } B(n-1) \geq K \\ 0 & \text{if } B(n-1) < K \end{cases} \quad (8)$$

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 delay, link propagation delay, switch processing delay, and transmission time of one arriving request). We assume that the feedback packets (ACKs) do not face any significant queuing delay, 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 suitable admission control is in place so that the *average* total offered traffic to the bottleneck link is always strictly less than 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 $W_i(t - \overline{R_i})$ in the last update interval $R_i(t - \overline{R_i})$, 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. 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 (first come first serve) scheduling for all packets.[2] 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, transferred data and throughput are in the units of packets rather than bytes. Based on the discussion above, we first restate the basic quantities below.

$$C_i(n) = \begin{cases} C.r_i/\sum_i r_i & \text{Throughput control} \\ \frac{W_i(n-1)}{R_i(n-1)} & \text{Latency Control} \end{cases} \quad (9)$$

$$Q_i(n) = \begin{cases} \frac{C_i(n)R_i(n-1)}{W_i(n-1)} & \text{Throughput control} \\ \frac{d_i + q_{ai}(n-1)/C_i(n-1)}{L_{it}} & \text{Latency Control} \end{cases} \quad (10)$$

$$\alpha_i(n) = \alpha_i(n-1) + \gamma[p(n) - \alpha_i(n-1)] \quad (11)$$

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

The order of calculation is as follows: We first estimate $p(n)$, i.e., whether ECN was received in the last update event, based on the queue length in the previous slot ($q_{ai}(n-1)$). This, in turn, is used to compute the fraction of BW given to each flow in the 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 packet transferred $W_i(n)$ and the update interval ($R_i(n)$) for the current slot. This is then used to estimate $q_{ai}(n)$, the queue length for the current slot, so that the temporal evolution can continue. Thus,

$$W_i(n) = \begin{cases} W_i + 1 & \alpha_i(n) \leq \varepsilon \\ W_i[1 - \frac{\alpha_i(n)}{2}] & \alpha_i(n) > \varepsilon \& Q_i(n) > 1 \\ W_i[1 - \frac{\alpha_i(n)}{2}]Q_i(n) & \alpha_i(n) > \varepsilon \& Q_i(n) \leq 1 \end{cases} \tag{12}$$

$$R_i(n) = 2d_i + B/C \tag{13}$$

$$q_{ai}(n) = \max[0, q_{ai} + W_i(n) - C_i R_i(n)] \tag{14}$$

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

One could seek the "steady state" from these equations by considering the case where the W_i and R_i do not change from slot to slot. However, it is clear from equation (12) 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 W_i 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 (12)). That is,

$$M = \inf_{(\sum_{m=1}^{M'} G_i^{(m)}) > R(n-1)} (M') \tag{15}$$

$$U_i(n) = U_i(n-1) - W_i'(n-1) + M - 1 \tag{16}$$

$$W'(n) = \min[W(n), U_i(n)] \tag{17}$$

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

B. Comparison of Model and Simulation

We validate the analytical modeling of QTCP with our ns3 implementation in Fig. 1 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 [10].

From Fig. 1 we can observe that our analytical model shows similar behavior in terms of throughput of the individual applications, as compared to the ns3 simulations. Thus this validates that our analytical model closely approximates the behavior 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.

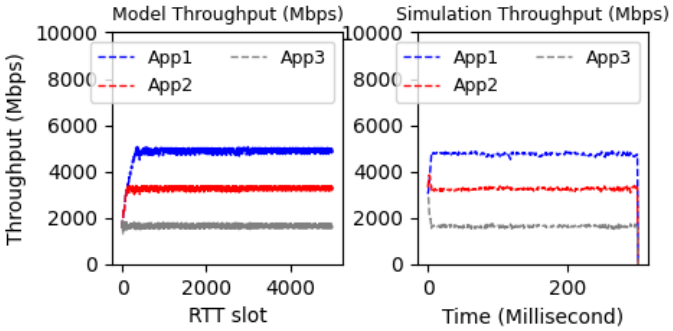

Fig. 1. Comparison of throughput between model and simulation

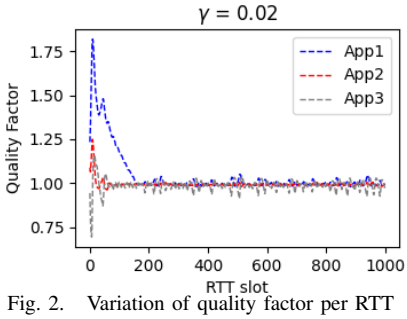

Fig. 2. Variation of quality factor per RTT

Fig. 2 shows the variation of Q_i with RTT slots, where the RTT and γ are assumed to be 248μs and 0.02 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. 3 shows the effect of different RTT values on the convergence time. As expected the convergence time increases with the increase in RTT values. In a data center environment, the RTT is relatively small (~150-200 μs) [15]; thus, the convergence time will be fairly quick. Fig. 3 also demonstrates that the quality factor of the applications will converge to 1.

IV. Performance Evaluation of QTCP

We comprehensively evaluate the QTCP mechanism using the popular ns3 network simulation package. For this, we

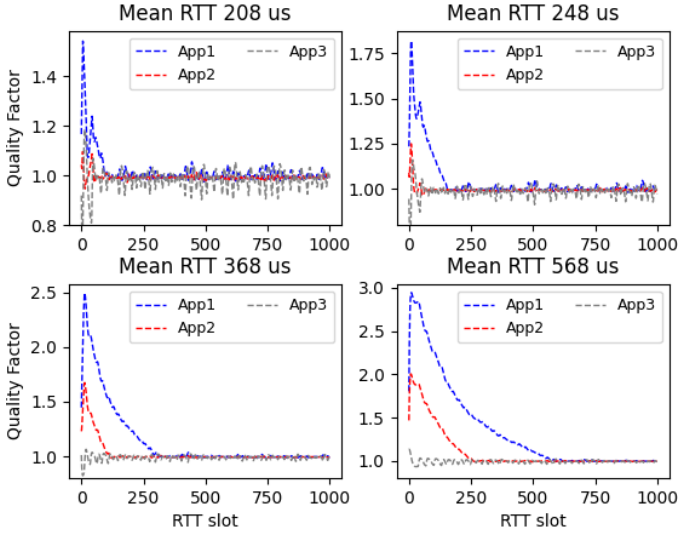

Fig. 3. QTCP convergence with different RTT

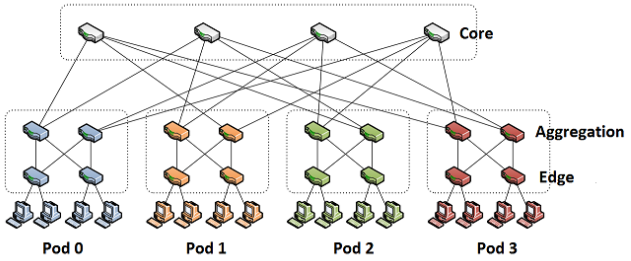

Fig. 4. Simulation setup; all links are connected through 10 Gbps port

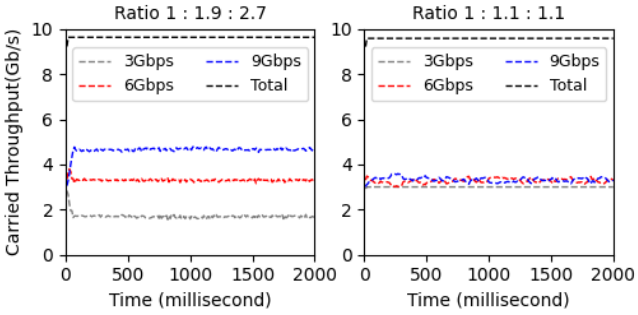

Fig. 5. Comparison of bandwidth distribution between QTCP (left) and DCTCP (right). The bottleneck bandwidth is kept as 10 Gbps. The attained throughput ratio of QTCP is approximately equal to the target throughput ratio, 1.0 : 2.0: 3.0.

started with the detailed DCTCP implementation in ns3 (which closely follows the RFC 8259) and also implemented the proposed QTCP mechanism. In spite of the existence of numerous data center network topologies, most data centers still use the fat-tree topology as illustrated in Fig. 4. Note that regardless of the size of network, the fat-tree topology always has 3 levels of switches (edge, aggregation, and core), which means that no request/response ever travels more than 6 hops. Thus the network in Fig. 4 is adequate for exhibiting QTCP performance, even though it is quite small. We used 10 Gb/s links for simulation efficiency, but similar results apply to the more contemporary 100 Gb/s links.

A. Bandwidth Distribution

In this section, we discuss the bandwidth distribution of applications having different QoS requirements. In this experiment, we have three 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. In the perspective of QoS aware Fabrics, we expect to distribute the available bandwidth in 1.0 : 2.0 : 3.0 ratio (i.e. 1.67, 3.33 and 5.0 Gbps respectively). Fig. 5 shows the comparison between QTCP and DCTCP in presence of 3 applications. In case of DCTCP, we can observe equal sharing of the bandwidth between the 3 applications in Fig. 5, 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, i.e. 9.63 Gbps as compared to 9.57 Gbps in case of DCTCP.

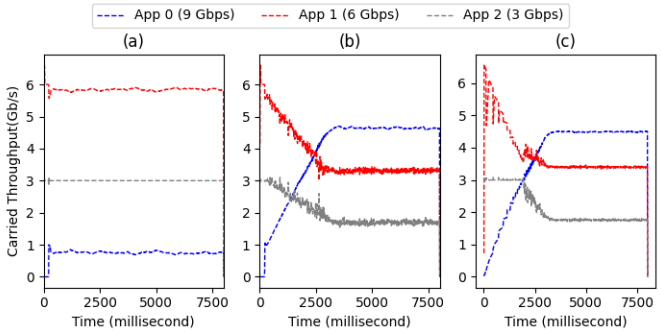

Fig. 6. Comparison of bandwidth distribution between (a) DCTCP and (b)-(c) QTCP for different RTTs. In (a) and (b), the RTT of 9Gbps application is higher as compared to other comparatively low priority applications. In (c), all the three applications are put into different fat tree PoDs. In both the cases, bandwidth distribution using QTCP is approximately equal to the target bandwidth ratio.

B. RTT fairness Comparison

We define RTT fairness as the extent to which we can achieve the target throughput ratios under different RTTs of the flows. For this, we generate intra-pod and inter-pod flows with different target ratios. Fig. 6 shows the results for both DCTCP and QTCP. 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. It is clear that DCTCP has a bias against flows with longer RTTs (Fig 6(a)), as flows with shorter RTTs grab bandwidth more quickly. One can argue placing higher QoS applications such that it experiences small RTT, but the problem is two fold: 1) This will cause other applications to starve, and 2) in a virtualized data center environment, this sounds impractical.

However, QTCP is not affected; it still offers the desired throughput ratios and the same overall throughput as DCTCP (9.6 Gbps). In QTCP, although the flows with shorter RTTs grab window more quickly, the quality factor forces the low QoS flows to make room for the high QoS flows, regardless of the RTT. Fig. 6 compares the results when a congestion occurs in several places, rather than only at the ToR (Top of the Rack) layer. Fig. 6(b) shows the result when the congestion is only at the ToR switch, whereas Fig. 6(c) shows the case of congestion in both aggregation and ToR layers. In both the cases, the bandwidth distribution ratio is close to the target throughput ratio.

C. DCTCP friendliness

The 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 explicit methods like [12]). If an interfering flow alters this value, each QTCP flow estimates it as shown in Fig. 8. Here target_i is the QoS requirement of flow i when there is congestion in the network, and actual_i is the estimated throughput till that point. Since quality factor Q_i quickly converges to close to 1 (section

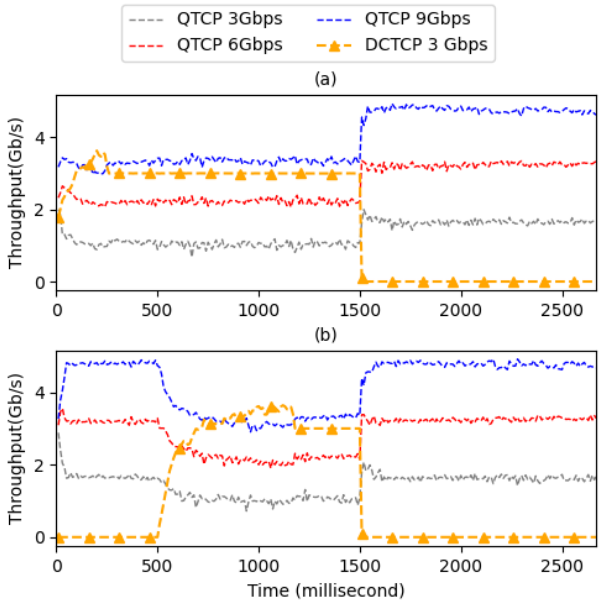

Fig. 7. Illustration of DCTCP friendliness of QTCP

III-C), its perturbation by more than σ is considered as a signal of a new interference or disappearance of the interference. We then change the target bandwidth target_i by the amount called "factor" and change the window size accordingly.

$W_i = W_i(1 - \alpha/2)$
if $(Q_i - 1.0 > \sigma)$) // means there is interference flow
factor = $-\sigma$;
else if $(Q_i - 1.0 < \sigma)$ //interference flow left
factor = σ;
else factor = 0.0
$\lambda = (\lambda + \text{factor}) \times \text{target}_i$
$\text{target}_i = \text{ratio}_i \times \lambda$; $Q_i = \text{target}_i/\text{actual}_i$
if $(Q_i < 1)$ $W_i = W_i Q_i$

Fig. 8. Window modulation in presence of interfering flows. $ratio_i$ defines the fraction of bottleneck bandwidth assigned), Q_i defines the current quality factor measured, W_i is the current congestion window size for flow i. λ is the bottleneck bandwidth measured/estimated. Also for our experiment we choose σ as 5% (i.e. 0.05).

Fig. 7 (a)shows the DCTCP friendliness results. When the DCTCP flow leaves (after about 1500 milliseconds of simulation time), the QTCP flows manage to grab the bandwidth resources according to the QoS specified ratio (1 : 2 : 3). In Fig. 7(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. We simulate with multiple DCTCP flows and get the expected result as 7(a) and 7(b) in all the cases. 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. Comparison of deadline misses 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; 3 out of 4 classes have latency requirements of 5366, 6604, 7832 μs respectively, whereas class 4 is best effort and has no QoS requirements. The mean transfer size we choose is 2MB. The flow arrival rate follows Poisson distribution and average utilization is 80%. Fig. 9 shows the comparison

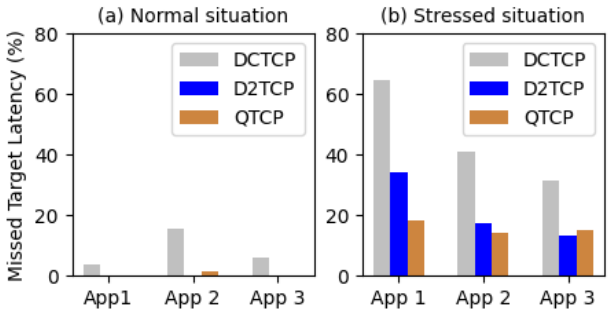

Fig. 9. Deadline misses in case of DCTCP, D^{2}TCP and QTCP under (a) normal and (b) stressed situations.

between QTCP and DCTCP for latency sensitive applications. In addition to that, we also simulate another QoS based window modulation scheme, named D^{2}TCP [17]. The D^{2}TCP proposal considers deadline of each class (or flow) while modulating the congestion window 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.

We simulate both the normal and stress situations to show the effectiveness of our scheme. Fig. 9(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 can meet their target latency, whereas in DCTCP ∼5-8% packets miss their deadlines. In Fig. 9(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% (i.e. ∼71% reduction as compared to DCTCP). Notice that in Fig. 9, we do not report the latency statistics for application 4, as it is of best effort type.

Fig. 9 also compares the performance of QTCP with D^{2}TCP. In the normal scenario, D^{2}TCP does not miss any deadlines. However, in stressed situation QTCP yields fewer 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.

V. QoS Differentiation in QRDMA

A. An overview of DCQCN

We now switch our gear towards RDMA transport mechanism, where DCQCN is considered as the popular transport protocol. DCQCN requires the PFC data center Ethernet feature, which is a L2 QoS capability to guarantee lossless transport for flows over a link. PFC cannot distinguish between traffic of same priority class, going to different destinations.

Thus the congestion along a given destination path will generate a PAUSE frame, which causes blocking of all the incoming flow packets belonging to the same priority class, even if the destination is different and there is no congestion at the corresponding path. DCQCN addresses these issues by leveraging both the PFC and ECN mechanism. However, the congestion feedback mechanism is different in DCQCN, since the underlying ROCEv2 utilizes connection less protocol i.e. UDP, so TCP like ACK feedback per frame transmission is not implemented. As a result, congestion feedback mechanism in ROCEv2 somehow needs to be request agnostic. Upon receiving a ECN marked packet, the receiver NIC sends ROCEv2 standardized congestion notification packet (CNP) to the sender NIC in N microsecond time interval, until the receiver NIC continues receiving ECN marked packets [18]. When the flow i sender gets a CNP, it reduces its current rate (RC_i) and updates the value of the rate reduction factor, α_i, like DCTCP, and remembers the current rate as target rate (RT_i) for later recovery, i.e.

$$RT_i(n) = RC_i(n-1) \tag{18}$$

$$RC_i(n) = RC_i(n-1)\Big(1 - \alpha_i \Big/ 2\Big) \tag{19}$$

The value of $RT_i(n)$ is used for flow rate increasing during the non congestion episode. Rate increase has two main phases; fast recovery, and additive increase. During the fast recovery phase, the flow rate RC_i is rapidly increased towards the fixed target rate for F successive iterations, as follows:

$$RC_i(n) = \Big\{RT_i(n-1) + RC_i(n-1)\Big\}\Big/2 \tag{20}$$

Fast recovery is followed by an additive increase, where the current flow rate slowly approaches to the target rate, and the target rate is increased by an additive constant R_{AI}, which is summarized as follows:

$$RT_i(n) = RT_i(n-1) + R_{AI} \tag{21}$$

$$RC_i(n) = \Big\{RT_i(n-1) + RC_i(n-1)\Big\}\Big/2 \tag{22}$$

B. QoS aware DCQCN - QRDMA

Similar to DCTCP, in case of DCQCN too the available bandwidth is distributed equally among all the existing flows. Thus, to implement QoS differentiation in RDMA, we develop the scheme QRDMA which essentially uses the same mechanism as developed for QTCP in section II.

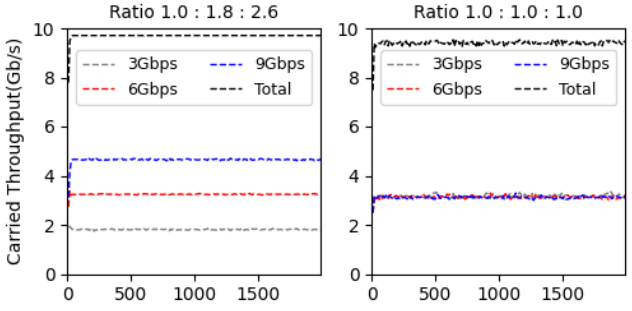

Fig. 10. Bandwidth Distribution in QRDMA (left) and DCQCN (right). Bottleneck bandwidth is 10 Gbps and attained carried throughput ratio of QRDMA is approximately equal to the target throughput ratio, 1.0 : 2.0: 3.0.

In particular, we control the target rates to provide the desired ratios for different classes. The analytic model in section III can also be applied to quantify the performance of various classes in QRDMA with some small changes, but we omit the details here due to lack of space. Fig. 10 shows the comparison between DCQCN and QRDMA with three applications; the simulation environment is exactly same as that of section IV-A. The attained throughput ratio of the three applications is 1.0: 1.8: 2.6, which is also approximately equal to the target throughput ratio (1.0: 2.0 :3.0). This shows that the QRDMA scheme also achieves expected service differentiation similar to QTCP.

Fig. 11 compares the performance of DCQCN and QRDMA with latency sensitive applications in a stressed scenario as in Fig. 9(b). From Fig. 11 we can observe that as compared to DCQCN, QRDMA demonstrates fewer deadline misses. In fact, as compared to DCQCN, QRDMA reduces the deadline misses by ∼43-80%. However, while comparing Fig. 9(b) and Fig. 11, we can observe that QRDMA yields higher deadline misses as compared to QTCP. This is mainly because the implementation of DCQCN (on which QRDMA is implemented) requires careful parameter tuning for the best performance. In our implementation of QRDMA, we have used the default parameter settings as suggested in [18]. Parameter tuning for QRDMA is beyond the scope of this paper and is left for future work.

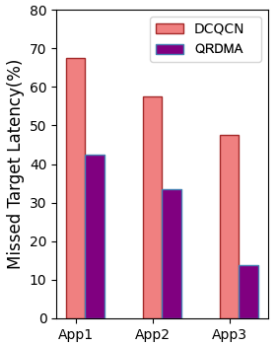

Fig. 11. Deadline misses in case of DCQCN and QRDMA

C. Effect of different incast degree

Single I/O request from a data center application might comprise of response flows from multiple storage servers. These multiple parallel flows can also be generated from complex interplay between data center applications. For example, in case of big data applications like Hadoop, or Spark, during the shuffle/reduce phase, several concurrent flows are generated. These concurrent independent flows can cause buffer overflow all of a sudden, and thus decrease application level throughput far below the link bandwidth. This phenomenon is known as *incast problem*. To avoid long latencies due to the buffer-bloat problem, the idea in DCTCP/DCQCN is to provision rather small packet buffers in the data center switches and control the traffic injection aggressively so the queues do not build up much beyond the point where ECN is triggered. However a switch queue to which a storage server is connected could still be overwhelmed due to incast problem and result in packet losses due to physical buffers being overrun. In this section, we discuss how a QRDMA works for different incast degree.

Fig. 12 shows the result for different incast degrees. In case of QRDMA, we observe a slight improvement (∼3%) in terms of the overall throughput. Since the endpoint has a notion of quality factor in QRDMA, it can modulate the data rate in a more controlled way, as compared to DCQCN. We also notice

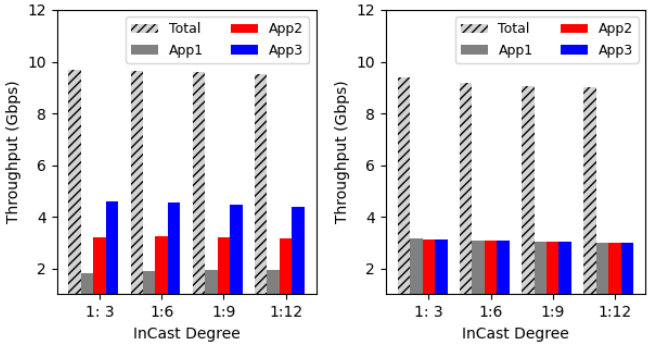

Fig. 12. Comparison of bandwidth distribution between QRDMA (left) and DCQCN (right) for different incast degrees.

that irrespective of the incast degree, QRDMA achieves QoS differentiation for different applications.

The incast issue can happen with TCP as well, and the experiments (not reported here) show we achieve similar QoS differentiation with QTCP as compared to DCTCP.

VI. Related Work

Although the main goal of conventional transport layer solutions (either TCP or ROCEv2) is fairness (equal sharing of bottleneck bandwidth) amongst different flows, some variants address the differentiated treatment. To the best of our knowledge, no work addresses the QoS issue in data center RDMA transport. But unlike QoS aware RDMA, several works have addressed the issue of QoS aware differentiated flow control in the TCP context. Homa [19], L^2DCT [20], D^2TCP [17], PDQ [21], D^3 [22] 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^2DCT and D^2TCP 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 the 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 [22] 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 transport solution for NVMe-oF that offers fairness by distributing the network bandwidth to different applications according to the relative QoS requirements. Our solution can be implemented using ECN-capable switches and does not need any specialized hardware. In this paper we considered QTCP and QRDMA separately, but it may be necessary to enable both simultaneously – QTCP for storage and QRDMA for persistent memory (PM). We shall examine this in the future. We shall also implement the two mechanisms in real servers and switches and explore a variety of storage and PM access scenarios. We shall also consider a mixture of classes with both throughput and latency related QoS requirements or a latency requirement along with a minimum committed rate requirement.

References

[1] "Nvm express over fabrics revision 1.1," https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1-2019.10.22-Ratified.pdf, 2019.

[2] Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan, "Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *ACM SYSTOR*, 2017.

[3] Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan, "Performance characterization of nvme-over-fabrics storage disaggregation," *ACM Trans. Storage*, 2018.

[4] M. Ray, J. Biswas, A. Pal, and K. Kant, "Adaptive data center network traffic management for distributed high speed storage," *IEEE LCN*, 2019.

[5] R. Davis, "The network is the new storage bottleneck," https://www.datanami.com/2016/11/10/network-new-storage-bottleneck/, 2016.

[6] "Addressing and data center bridging (dcb)," https://1.ieee802.org/dcb/.

[7] F. Baker, D. L. Black, K. Nichols, and S. L. Blake, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, 1998. [Online]. Available: https://rfc-editor.org/rfc/rfc2474.txt

[8] J. Polk, F. Baker, and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic," RFC 5865, 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5865.txt

[9] S. Scargall, "Remote persistent memory," in *Programming Persistent Memory*. Springer, 2020, pp. 347–371.

[10] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Pate, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010.

[11] "Infiniband™ architecture specification release 1.2.1 annex a17: Rocev2," https://cw.infinibandta.org/document/dl/7781, 2014.

[12] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[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, W. Bai, G. Chen, K. Chen, D. Han, and Y. Zhu, "Combining ecn and rtt for datacenter transport," in *APNet*, 2017.

[16] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *ACM SIGCOMM*, 2016.

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

[18] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *ACM SIGCOMM*, 2015.

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

[20] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *IEEE INFOCOM*, 2013.

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

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