Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers

Prajjwal Devkota
Department of Electrical and Computer Engineering
Texas A&M University
College Station, Texas 77843
Email: prajjwal@tamu.edu

A. L. Narasimha Reddy
Department of Electrical and Computer Engineering
Texas A&M University
College Station, Texas 77843
Email: reddy@ece.tamu.edu

***Abstract*—This paper analyzes the performance of Ethernet layer congestion control mechanism Quantized Congestion Notification (QCN) during data access from clustered servers in data centers. We analyze the reasons why QCN does not perform adequately in these situations and propose several modifications to the protocol to improve its performance in these scenarios. We trace the causes of QCN performance degradation to flow rate variability, and show that adaptive sampling at the switch and adaptive self-increase of flow rates at the rate limiter improve performance in a TCP Incast setup significantly. We compare the performance of QCN against TCP modifications in a heterogeneous environment, and show that modifications to QCN yield better performance.**

I. Introduction

Data is being stored in big data centers and accessed increasingly over wide area networks. Google, Yahoo, IBM, Microsoft, Amazon.com and many others are providing services to store, access and process data in such data centers. The economies of scale are driving these data centers to become big.

The data centers are typically organized with many storage devices, associated servers to manage the data and Ethernet switches to interconnect these servers within the data centers. The data of one user may be spread across or striped across many servers for performance or reliability reasons. Distributed filesystems such as Panasas [1], NFSv4.1 [2], lustre [3] and distributed search queries, such as employed by Yahoo! [4] are becoming increasingly popular. When a user accesses data from data centers, the data will cross the data center Ethernet switches during the data delivery to user. The Ethernet switches, typically, have small buffers of the range 32KB - 256KB and these small buffers may overflow at the times of congestion.

Data is accessed over wide area networks using network transport protocols such as TCP and UDP. When a packet is dropped in the Ethernet switches due to congestion, TCP automatically adjusts its rate using a closed loop Additive Increase Multiplicative Decrease [5] algorithm. A transport protocol such as UDP, on the other hand, does not do any rate limiting by itself, necessitating that the layers above implement congestion control. Other transport protocols such as Real-time Transport Protocol [6], Structured Stream Transport (SST) [7] may be used by different applications.

II. TCP Incast

In a clustered setup, data is distributed across multiple servers for increasing performance and reliability, with a client relying on several servers to send parts of the data to it in parallel. Dividing data across multiple servers is generally a good solution to decrease data access latency as well as increase reliability of the storage system. However, it has been found that increasing the number of servers beyond a certain point results in a catastrophic drop in the network throughput [8]. This problem is known as the TCP Incast problem. Please refer to Figure 2 to see an example of TCP Incast occurring in our simulations.

This drop in network throughput arises from the interplay of heavy congestion (and hence packet drops), TCP rate control and timeout mechanisms, and the synchronized nature of the traffic. The synchronized nature of the traffic implies that if one flow slows down for some reason, all flows must slow down to the same rate to allow the slow flow to catch up. Packet drops due to heavy congestion cause TCP timeout mechanism to kick in [8]), and the minimum period for which a transmission must wait is 200ms: a reasonable value in most networks, but a very long interval in the high-bandwidth, low latency networks of datacenter networks. This causes the timed-out flow to considerably slow down, while all other flows must wait for it. Effectively, all flows are thus slowed down, leading to drastic throughput collapse. The preconditions for TCP Incast are listed in [9], as: High-bandwidth, low-latency networks, Clients issuing barrier synchronized requests (i.e. a new request is not issued till all responses to current request have been received), and servers returning data in small chunks that cannot saturate the link for long.

Phanishayee et. al. have proposed reducing the TCP minimum retransmission timer (minrto) from its current value of 200ms to values of 1ms or lower in [10], and show that using a small TCP minrto value does not negatively affect TCP performance in general scenarios. A different approach is examined in [11], where the authors investigate Incast occurrence in a datacenter environment, and propose a modified TCP which aggressively maintains a low switch queue length in order to keep enough buffer to deal with sudden bursts of traffic while maintaining very low queuing delays.

III. Quantized Congestion Notification

In a datacenter environment, TCP may not be the only protocol used, and multi-protocol environments are possible. Some video delivery and streaming applications employ UDP and NFS servers can also run on top of UDP. Transport protocols such as UDP do not employ congestion control, and rely on other layers to do the rate adjustments. Standards are currently being developed to implement flow control at the Ethernet link layer. These standards aim to allow fair sharing of bandwidth among multiple flows which may or may not employ rate control mechanisms themselves, and to allow datacenter applications requiring lossless and very low latency environments to function effectively. It is very likely in the near future to see datacenter switches implementing these protocols become common.

The IEEE 802.1 QAU Congestion Notification standards specify protocols, procedures, and objectives for congestion management of long-lived data flows in low bandwidth delay product networks [12]. The aim of these standards is to provide support for loss and latency sensitive higher layer protocols, without requiring any rate controlling mechanism in the higher protocols. The protocols are employed at the link layer, and are effective only within the **congestion management domain**, which consists of the switches and hosts which support the Qau protocol. The switches employed are able to send congestion information to traffic sources, which are able to rate limit their flows to avoid frame loss. Currently, the standard being examined the most is Quantized Congestion Notification (QCN). While other clusters also use various congestion control algorithms [13], [14], QCN aims to provide congestion control for datacenters where Internet Protocol Suite protocols such as TCP and UDP are running.

QCN consists of two main components: the Congestion Point (CP) and the Reaction Point (RP). When transmitting data from a traffic source to its destination host, all intermediate switches can become congested, and are hence called congestion points. These switches regularly sample packets and monitor their queue lengths to make sure congestion is not occuring, based on the growth rate of the queue and its offset from an equilibirium point QEQ, around which the queue length is intended to stabilize. A CP calculates a feedback value based on the queue state, and if the computed feedback is negative (implying congestion), sends this feedback to the source of the sampled packet. The feedback, however, is quantized into a typically 6 bit value (between 0 and 63), leading to the name Quantized Congestion Notification.

The RP, upon receiving negative feedback, multiplicatively decreases its rate. However, it cannot increase its rate based on feedback, as there is no positive feedback. QCN rate increase consists of three stages: i. Fast Recovery(FR), ii. Active Probing (AI) iii. HAI. The aim of FR is to rapidly get back to the rate at which feedback was received if the congestion was transient. In AI stage, small increments are made in bandwidth to probe link capacity. This is the stage where QCN is stable around QEQ. HAI is a much more active probing stage, where rate is increased rapidly each cycle, with the assumption that lack of negative feedback implies the link is underutilized. For details of QCN, refer to [12].

QCN and changes to TCP are two different approaches to solving the congestion problem in data centers. QCN implements congestion control in the Ethernet link layer and potentially allows applications with different transport protocols (TCP and UDP) to co-adjust in the data center. QCN rate control is equivalent to physical bandwidth restriction: the flow can use a maximum rate not exceeding whatever QCN has allocated to it at the time. TCP flow control, based on controlling the sender's window size controls the sending rate, but based on whatever physical bandwidth is available at the time of sending, and is hard limited by the QCN assigned rate. In our simulation setup, only one flow is present on a link in any direction, and hence, we will be referring to flow rate as link rate to make it easier to intuitively understand: the QCN link rate will be the maximum bandwidth physically usable by the flow.

Changes to TCP, such as modifications to timers, are specific to TCP, and address TCP's problems in datacenter scenarios without requiring extensive modifications to data center network infrastructure. As we will point out later, QCN tries to avoid packet drops and consequent TCP timeouts that result in performance loss, while modifications to TCP timers recover quickly from TCP timeouts without necessarily reducing the number of packet drops.

In the rest of the paper we study if QCN is effective in TCP Incast scenarios. Our results show that QCN is not very effective in controlling the congestion collapse in the Incast scenarios. We analyze the causes of this and propose two modifications to QCN to improve its performance in TCP Incast scenarios. We show that QCN with the proposed enhancements can be an effective solution in mixed workloads with TCP and UDP in datacenters.

IV. TCP Performance in Incast Scenario without QCN

Figure 1 shows the basic simulation setup, similar to what is used in [9]. There is one single client requesting data blocks from n servers, each of which returns a fixed size block in response to a request sent by the client. The client waits for server responses, and when it has received all of the responses, it sends the next request to the servers. All links are 1 Gbps links, and when the servers send their responses back, the switch –> client link gets saturated. This leads to packet drops, and with a sufficient number of servers, TCP Incast occurs.

The response each server sends when the client sends a request is known as a Server Response Unit (SRU). Until a client has received SRUs from all servers, it will not issue the next request. There is a single TCP connection between each server and the client during the simulation, which will be referred to as a flow. When QCN is employed, QCN controls the rates of these flows, but as only one flow is active per link, the paper will refer to these rates as link rates.

All simulations have been done using the ns-2 network simulator. Unless otherwise specified, all simulations have a runtime of 20s, and the server response units are 256KB each. Switch buffer sizes of 32, 64, 128, and 256KB have been used, and flows upto 64 have been used, but with only flows that are powers of 2 (2,4,8,...).

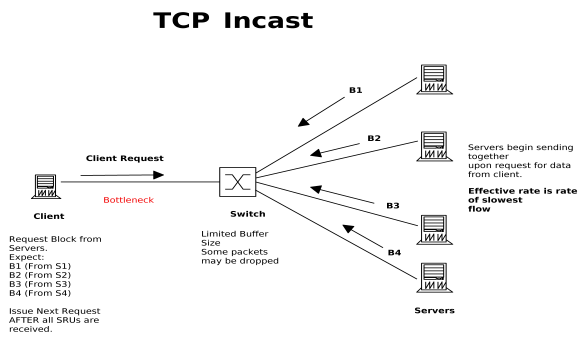


Fig. 1. Simulation Setup for TCP Incast Collapse

A. Unmodified Newreno

We first verified the occurrence of TCP Incast with unmodified newreno, and link layer congestion control mechanism. The link utilization versus the number of flows for 32, 64, 128, and 256 KB buffer sizes was plotted, and incast collapse was seen to occur as early as with 8 flows for 32 KB buffer size, collapse occurred at 32 flows for 256KB switch buffer size. The performance of unmodified newreno is shown in Figure 2. This was found to be in accordance with the results presented in [10].

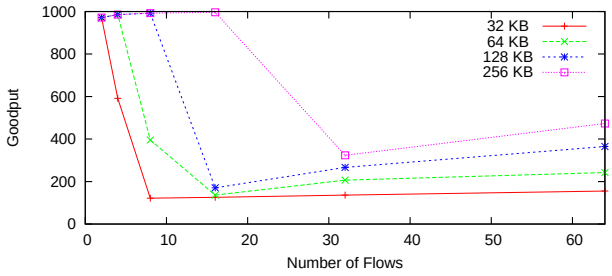


Fig. 2. Unmodified TCP NewReno Goodput vs. Number of Flows

B. Newreno with Reduced MinRTO

We reduced the minimum retransmission timeout for TCP, and ran simulations with rtos of 10ms and 1ms, as suggested in [15]. The performance of TCP with a minRTO changed to 10ms is shown in Figure 3. As seen in Figure 3, performance is much better, and there is almost 100% link utilization for 256KB buffer sizes, while goodput drops to slightly above 60% for 32 and 70% for 64 KB buffer sizes. The reason for improved performance is that, even though timeouts themselves are not prevented, the effect timeouts have on the entire flow rate is minimized to a negligible amount [15].

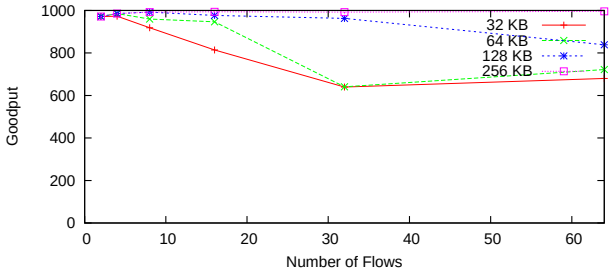


Fig. 3. TCP with 10ms minRTO Goodput vs. Number of Flows

V. QCN Performance in TCP Incast

QCN can effectively control link rates very rapidly in a datacenter environment. However, we find that it performs rather poorly in the context of the TCP Incast scenario, due to the synchronized nature of the Incast flows. The performance of TCP incast with QCN is shown in Figure 4. The average queue lengths over the simulation period (not shown in figures) is over 200KB with 256KB buffer size and only newreno with 10ms rto at 64 flows. With QCN however, the average queue length is only 70KB with 256KB buffer size. Queue length for fewer flows are still high for 256KB buffer size and newreno with 10ms rto: the average queue length is 140KB at 8 flows, and grows with more flows. The average queue length of QCN is 20KB for 8 flows, and grows upto 70KB at 64 flows. This indicates that QCN does keep queue lengths controlled, but as seen from the figure, the performance in an incast setup is not good. As we shall explain why soon, we also present the results of TCP with 10ms minRTO with unmodified QCN at the link layer in Figure 5.

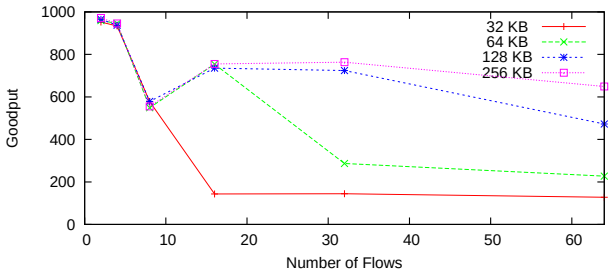


Fig. 4. Unmodified NewReno TCP over Unmodified QCN Goodput vs. Number of Flows

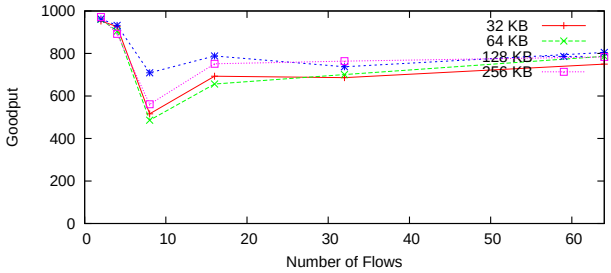


Fig. 5. 10ms minRTO TCP over Unmodified QCN Goodput vs. Number of Flows

We make several observations from the results in Figures 4,

5 immediately: i. QCN does not achieve full link utilization ii. after collapse (throughput below 500Mbps), QCN goodput is similar to that of only unmodified newreno at smaller buffer sizes iii. before collapse(throughput above 500Mbps), the performance of QCN, with or without modified TCP (reduced minimum retransmission timer) are quite similar and iv. QCN provides higher goodput with higher buffers than when QCN is not employed after unmodified TCP collapses due to incast. We shall deal with these observations in the coming section. In later sections, we propose modifications to basic QCN to make it perform well in the Incast scenario.

VI. Analysis of QCN Performance

First, we need to point out that there are several releases of the QCN pseudocode. Our implementation of QCN is as per QCN pseudocode version 2.2 released by Rong Pan [16] [1]. We shall be referring to our implementation of QCN based on the latest available release of the pseudocode ([16]) as unmodified QCN, and all proposed modifications are applied to this implementation.

First, let us consider QCN performance without TCP time-outs, or with very little penalty of timeouts (using TCP with reduced minrto). TCP performance before timeouts is very good, utilizing 100% link capacity, and modified TCP, with reduced penalty of timeouts, utilizes nearly 100% link capacity as well. This suggests that link utilization should be near 100% with QCN at the Ethernet Link Layer and the same transport layer protocols above it. However, with QCN enabled, full link utilization is not achieved. In fact, in Figure 5 we see that when there are more than 8 flows, even with TCP with minrto reduced to 10ms, link utilization does not go above 80% using QCN at the link layer, and drops to 45% link utilization with 8 flows.

To explain this low utilization, let us refer to a simulation with 128KB buffer size and 8 flows. We first look at Figure 6, where we see that the minimum link rate is constantly below 80 Mbps throughout the simulation. However, as seen in Figure 7, where 3 flows are examined, we see that the rate allocated to each flow varies with time, and the flow with minimum rate is not fixed over time.

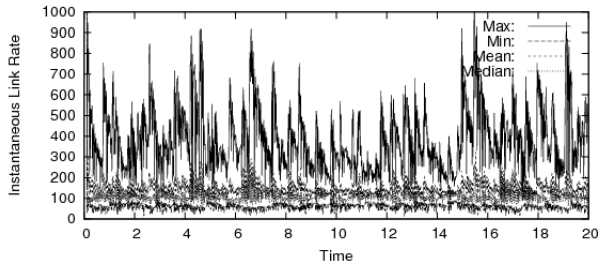


Fig. 6. Link Rate Statistics for QCN: 8 Flows 128KB Buffer Size

[1]This version of pseudocode employs slightly different sampling as compared to the previously released version [17]. The effects of this in both incast and baselines are noticeable though with the older pseudocode performing better than version 2.2 in Incast.

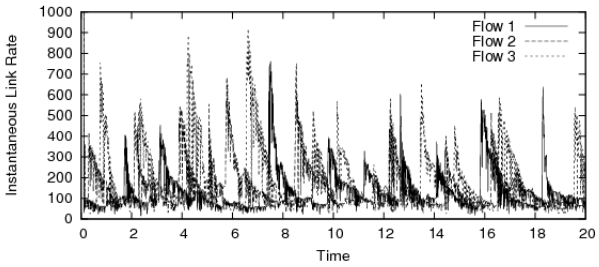
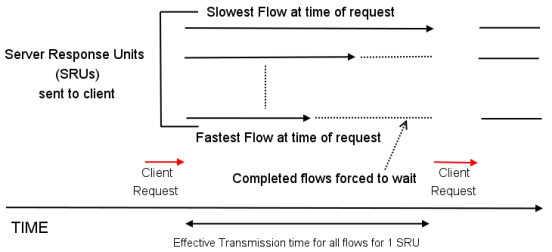


Fig. 7. Link Rates for 3 Flows for QCN: 8 Flows 128KB Buffer Size



Effect of Link Rate Variation on Barrier Synchronized Flows

Fig. 8. Effect of Link Rate Variation on Barrier Synchronized Flows

Consider the time it takes for QCN to increase and decrease link rates. Rate decrease is instantaneous: each time negative feedback is received, the link rate is reduced multiplicatively. However, link rate increase only takes place if negative feedback has not been received for a certain period. This period can either be 15ms (timer based), or the time to transmit 150KB (byte counter based). These periods are sufficiently fine-grained for most purposes, but let us take a closer look at the timescales involved in our simulation.

A client request for data causes each server to send back a Server Response Unit (SRU), which is 256KB in our simulations. The time to transmit 256KB using the full 1Gbps link capacity is roughly 2 milliseconds. However, if we assume that the flow can physically only use 100Mbps because of QCN rate limiting, then the time to transmit one SRU is roughly 20 milliseconds. The timer and byte counter based thresholds described in the preceding paragraph do not take much effect during the time to transmit a single SRU, implying that not much rate increase occurs during this period. Hence, we can approximately assume that link rates are static during the time to transmit a single SRU. This further implies, in our example simulation of 128KB buffer size and 8 flows, that the minimum link rate while transmitting an SRU is 80Mbps or less. No matter what the maximum link rate is, the effective rate of all flows thus becomes 80Mbps or less due to flow synchronization at the SRU granularity, leading to an effective throughput of less than 8 x 80=640Mbps.

The phenomenon of the smallest link rate becoming the effective link rate is illustrated in Figure 8. The minimum rate among all of the link rates at each sampling interval during the simulation is termed the minimum link rate for that interval. The average value of the minimum link rate across

the simulation period is referred to as the average minrate.

If our hypothesis that the effective rate for all the flows is roughly equal to the minimum rate at any given time, then the averaged value of the minrate x the number of flows should give the effective goodput for the simulation before TCP collapse occurs. This is corroborated by the data in Table I. Though we have not shown data for other simulations for brevity, this relation holds true before the onset of TCP timeouts in all QCN based simulations (as well as those with TCP with reduced minrto over QCN, since timeout penalties are very small in this case).

TABLE I
AVERAGE MINRATE AND GOODPUT FOR OUR IMPLEMENTATION OF QCN FOR: 128KB BUFFER SIZE

Flows	Avg Minrate	Avg Minrate x Flows	Goodput	Avg. Timeouts
2	512.19	1024.37	966.61	0
4	208.74	834.94	835.29	0
8	59.31	474.46	484.77	0
16	36.21	579.312	613.37	0.063
32	29.4	**940.77**	**325.01**	8.94
64	32.03	**2049.8**	**408.4**	22.75

The bold entries in the table indicate the point at which the goodput becomes much less than the sum of link rates (avg minrate x flows) of all the flows. It is observed at this point, QCN is unable to control congestion and timeouts start occuring, and the number of timeouts increase as with the number of flows.

It can be seen from Table I that when TCP timeouts begin occurring (last column corresponds to TCP timeouts per flow), average minrate x number of flows is much greater than the observed goodput of the flows. This happens because of the TCP timeouts: even though a flow is allocated a certain link rate, it cannot utilize that rate if it has to wait for at least 200ms before it can start sending again. Hence, the effective rate of the timed out flow is very low, and this low rate affects all other flows as they are all syncrhonized, leading to a very low utilization throughout. This effect is illustrated in Figure 8, where the faster flows stay idle until the slowest flow at the time of the request completes its transmission and the client issues a new request.

During the 200ms interval when a flow is waiting for a TCP timeout to expire, the other flows are waiting for a client request. During this time, the flows do not receive any negative feedback from QCN CPs. Lack of negative feedback is perceived as no congestion, and the RPs increase the available link rates for flows by an amount controlled by the parameter R_AI (which is set to 5Mbps) during FR, and R_HAI (50Mbps) during HAI. As pointed out earlier, link rate can be increased every 15ms or after successfully sending 150KB of data. The 200ms timeout interval corresponds to roughly 13 15ms intervals, and hence 13x5=65Mb rate increase could occur in the link rate (if in the AI region of QCN). During the HAI stage, this figure is 10 times as much. When the TCP timeout period expires, all the flows could have seen a 65Mb rate increase, and with even 8 flows, this is equivalent to half the total link capacity. When the flows begin transmission again, this increased link rates leads to congestion again and timeouts occur, and so on. This can be observed in Table I, where the avg. min rate x num flows exceeds the link capacity of 1 Gb for 64 flows.

In order to avoid this problem (not enough negative feedback during severe congestion and subsequent significant increase in link rates), we propose two modifications to QCN, in the next section. The first solution effectively increases sampling rate during severe congestion and hence increases the rate of feedback. The second solution adapts the link rate increase (5Mb in earlier discussion) to adapt to the number of flows and hence tries to control the link rate increase bursts.

One possible solution to improve fairness at small timescales is to employ fair queuing at the Ethernet switch. Many variants of fair queuing [18] are available with different levels of fairness at small time scales. We do not explore this solution here because of potential increase in complexity of implementation at high speeds.

VII. MODIFICATIONS TO QCN

A. Sampling Modifications

In order to test our hypothesis of lack of sufficient feedback during severe congestion phases, we tried to see if sampling every packet would make any difference. The results were much better, and collapse did not occur even for simulations with 32KB buffer size. Since every packet was sampled, the number of QCN feedback packets sent increased. However, this number was just around 5 times as much as normal sampling, as only negative feedback packets are sent in QCN, and not all sampled packets generate negative feedback. However, as sampling every packet might not be necessary, especially during periods of less congestion, we propose two different strategies which offer almost the same level of performance as sampling every packet, while requiring fewer packets to be sampled during periods without congestion. These strategies are described in the following sections.

1) Congestion Memory Based Sampling: During a TCP timeout, there is little or no traffic through the bottleneck. No traffic implies that the RPs are not transmitting, and this means that they do not receive negative feedback. This lack of negative feedback causes them to ramp up their rates during the timeout period. On the other hand, the CP sees that its queue is empty, and tends to sample fewer packets according to the QCN sampling algorithm. However, as soon as the victim flows complete their remaining transmissions, all flows will send data at high rates, causing incast to occur again. If the CP actually sampled more packets during a period of timeout, anticipating a flood of traffic, this problem could possibly be alleviated.

The only way to distinguish between genuine lack of traffic and lack of traffic due to a 'period of silence' being in progress is by keeping track of previous congestion events: if heavy congestion had recently occurred, then a period of silence is probably a result of congestion. This means that when traffic resumes, sampling should actually be more, and not

less. For this reason, a memory variable (simple counter) is added to the CP, and is incremented every time a congestion event occurs. We defined a congestion event in terms of queue length exceeding 90% of the queue, though using feedback could possibly be more appropriate in terms of the current implementation of QCN. We scale the sampling frequency exponentially based on this counter (multiplied by 2 to the power of this counter value). In terms of implementation, this strategy requires an extra memory element to keep track of congestion events, and also needs logic to divide the sampling interval by the memory element. However, the dividing logic consists simply of right shifts equivalent to the value of the counter, as the divisions are in powers of two. Hence, the extra overhead to implement this strategy in terms of memory and computation is not very high. We also need to define an extra variable equivalent to 90% of the queue (QSC), but using the calculated feedback value as an indicator of congestion should be as effective.

The performance of this strategy in the incast setup is shown in Figure 9. The performance is considerably improved compared to the QCN results shown earlier in Figure 4.

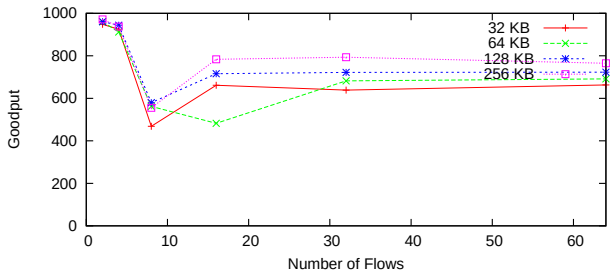


Fig. 9. Congestion Based Memory Sampling in Incast Setup

2) Detailed Sampling: A closer look at QCN performance for flows that do not cause collapse shows that the link rate converges very rapidly. After collapse, however, this convergence does not occur as TCP flows time out before convergence, leading to spurious increases in QCN flow rates.

However, with every packet being sampled, flow rates converge even when normal QCN collapses. This implies that more frequent sampling during heavy congestion than what is employed by normal QCN will be helpful. QCN quantizes feedback in terms of 64 levels (0-63), but the sampling at the CP is quantized into 8 levels i.e., only 8 different sampling frequencies are employed based on the congestion level. We expanded the heaviest congestion level into eight more sampling frequency levels to allow more frequent feedback. This leaves the sampling for less severe congestion unchanged, while sampling during periods of heavier congestion is stepped up aggressively.

In terms of implementation cost, no extra memory resources are required, and a slight change in the sampling algorithm with a sampling lookup table that has 15 levels instead of 8 levels is used. Hence, this is very easy to implement in hardware, and requires no extra logic or changes to other elements elsewhere in the network. The performance of this strategy in the incast setup is shown in Figure 10. The results are observed to be considerably better than the QCN results shown earlier in Figure 4.

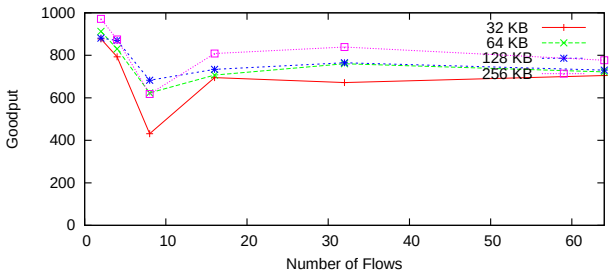


Fig. 10. Detailed Sampling in Incast Setup

B. Adaptive R_AI

As observed in the QCN analysis section, QCN performance in TCP Incast is not very good because of the difference between minimum and maximum flow rates at a given instance of time. The average values of the flows do converge around an optimal division of rates, but the fluctuation of rates in the Active Probing stage of QCN creates enough variations that the variation in flow rates causes sub-optimal throughput for synchronized flows.

We reduced the amount by which an RP increases its rate during congestion by making the self-increase rate (R_AI) congestion aware. This was done by setting R_AI 5Mbps when no congestion occurred, but a fraction of this amount with negative feedback. The adjustment is done by dividing R_AI by a memory element which keeps track of received congestion feedback. As feedback can be any value between 0 and 63, with 63 signifying the highest level of congestion, the memory element keeps track of the feedback severity as well: the absolute value of the feedback level is added to the memory element, and this value divided by 64 (to normalize it) is used to divide R_AI. The same strategy is used on R_HAI too, if the self-increase is in the HAI phase.

The memory element is decremented by 1 each time the RP timer expires, leading to a slow increase in R_AI rate with recovery from congestion. In terms of implementation cost, an extra counter is required in every Reaction Point. Further, logic to divide R_AI and R_HAI by the memory element needs to be added, but this can be replaced by a lookup table if necessary, as the possible values of effective R_AI are always 5Mbps (or whatever value of R_AI is implemented) divided by fixed increments of a bounded counter. With the use of a lookup table, the computational overhead is low, while some memory needs to be added for the table itself, but should not be very expensive, as it is on the RP.

The performance of this strategy in the incast setup is shown in Figure 11. The results are observed to be considerably better than the QCN results shown earlier in Figure 4, though performance still collapses past 32 flows for 32KB and 64 KB buffer sizes.

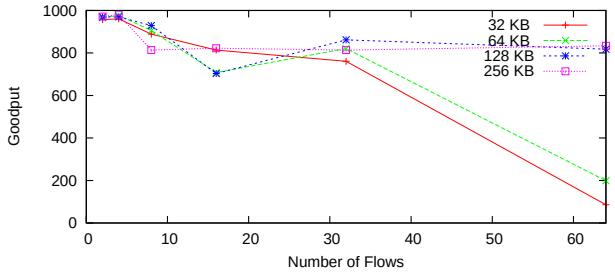


Fig. 11. Adaptive R_AI in Incast Setup

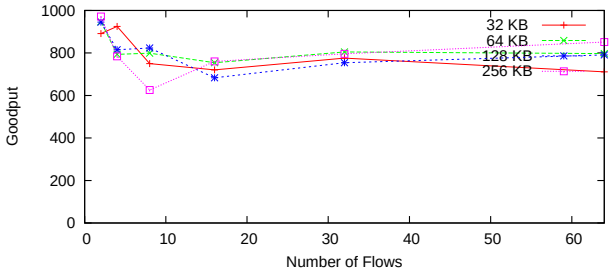


Fig. 13. Adaptive R_AI with Detailed Sampling in Incast Setup

TABLE II
QCN FEEDBACK PACKETS GENERATED FOR 128KB BUFFER SIZE

Flows	I	II	III	IV	V	VI
2	4740	7282	11716	3357	3337	8905
4	5535	10514	10299	4454	5920	4853
8	6530	10613	15386	7879	8762	8687
16	8844	15494	19894	10840	12150	13444
32	14551	30779	42621	21912	20146	28112
64	21846	156161	121382	54405	36076	62553

I: Unmodified QCN II: Congestion memory based sampling III: Detailed Sampling IV: Adaptive R_AI V: II + IV VI: III + IV

C. Combining CP and RP modifications

We also ran simulations with both the CP side (sampling) modifications and RP side (R_AI) modifications implemented. This was done for both of the sampling strategies described previously.

The performance graphs of combining adaptive R_AI, which is an RP side modification, with sampling modifications, which are CP side modifications, are shown in Figures 17 and 18. As can be seen, a combination of the RP and CP side modifications show even better performance results than either of the strategies alone. An intuitive explanation of this is that while RP modifications (adaptive R_AI) reduce aggressiveness of rate growth during periods of lesser congestion, the CP modifications (detailed sampling and congestion memory based sampling) act during periods of high congestion by notifying the RPs more rapidly that they are causing congestion. As they work in different phases of the incast cycle, combining them has a better effect on the overall performance of QCN in a TCP Incast setup.

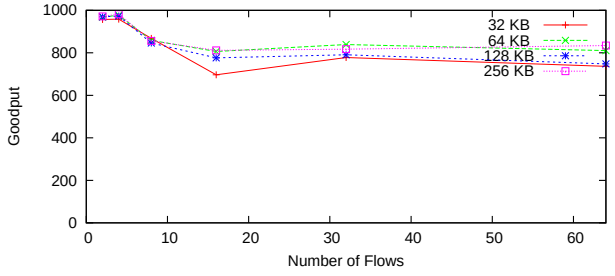


Fig. 12. Adaptive R_AI with Congestion Memory Based Sampling in Incast Setup

Further, the average queue lengths over the simulation periods are even lower than that with unmodified QCN, with the average queue length constantly below 30KB even with 256KB buffer size, upto 64 flows with both proposed schemes.

While we did not examine the effects on request latency in the simple setup examined, a low queue length can help mitigate incast effects due to timed out requests as well as prevent drops due to bursts of traffic by keeping buffer space free [11].

The number of feedback messages sent in various systems with different modifications to QCN are shown in Table II.

It is observed that the proposed modifications increase the number of feedback messages sent. As observed earlier, during heavy congestion events, higher sampling is desired. It is observed that adaptive R_AI technique and its combination with sampling techniques require significantly fewer number of feedback messages than when sampling techniques are employed by themselves.

TABLE III
QCN SIMULATION TIMEOUTS:128KB

Flows	I	II	III	IV	V	VI
16	0.12	0	0	0.06	0	0
32	0.44	0	0	0.96	0	0
64	9.28	0.22	0.28	4.31	0.21	0.22

I: Unmodified QCN with modified TCP II: Congestion memory based sampling III: Detailed Sampling IV: Adaptive R_AI V: II + IV VI: III + IV

The average number of timeouts experienced by each flow in various experiments are shown in Table III. It is observed that higher sampling is more effective in controlling the number of timeouts and timeouts can be kept very low, even with a high number of flows using faster sampling. The results also show that the significant number of timeouts occur with only changes to RTO timers of TCP.

VIII. PERFORMANCE WITH EXTRA FLOWS

As we saw in section VII-C, unmodified TCP also performs quite well with QCN that has both CP and RP side modifications implemented. However, link utilization is not 100%, with it falling down to as far as 80% for 256Kb buffer sizes and 70% for 32Kb buffer sizes for both strategies for upto 64 flows (Figures 17 and 18). However, as can be seen in Figure 3, the performance of TCP with modified minRTO is much better: with almost 100% link utilization for 256KB buffer size, and lesser utilization for lower buffer sizes, even with 10ms TCP minrto.

However, QCN will most likely be implemented soon in datacenter switches, and any performance problems due to the switches implementing QCN will lead to the performance issues that we saw in section VI. Further, it is quite possible that a clustered storage setup will not only have TCP flows from the incast setup, but will also have UDP flows and possibly other protocols in the mix as well. At least in the case of UDP, TCP tends to yield most of the bandwidth demanded by an aggressive UDP protocol. The aim of QCN is to function in such an environment, and we ran simulations in a mixed protocol environment as well.

We modified our simulation setup to add one high bandwidth UDP CBR flow (400Mbps) from an additional node to the client node, and ran the simulations both with and without QCN, and both with and without a modified (reduced minRTO) TCP. TCP performance (Figure 14) undergoes collapse with half the number of flows than in the simple setup. Our results show that the setup with modified TCP with minrto 10ms only (no QCN) yielded 400Mbps to the UDP flow (Figure 15), and fairly utilized the remaining bandwidth with 256KB buffer size. The performance was slightly worse with lower buffer sizes, but we believe this performance would go upto nearly 60% link utilization with smaller minrto values as proposed in [15].

However, with a modified QCN (Figures 17 and 18), the total bandwidth utilized by the synchronized requests was around 800Mbps for 64, 128, and 256 KB buffer sizes, and around 700Mbps for 32 Kb buffer size, past 8 flows. 256KB and 128KB buffer sizes did not perform so well at 4 and 8 flows, dropping upto 400Mbps utilzation. The performance of unmodified QCN (Figure 16) was not so good, though it was still better than that of modified TCP for 256Kb buffer sizes in flows except 8 flows, where it drops to 500Mbs bandwidth utilization.

The performance of modified QCN with both modified and unmodified versions of TCP running on top of it are very similar, so we have only shown the performance of unmodified TCP above QCN with the proposed modifications. The queue lengths with newreno with 10ms rto and 256KB buffer size is constantly above 200KB, while with combined CP and RP modifications, the queue length is constantly below 40KB upto 64 flows, and with detailed sampling employed at the CP, actually stays below 20KB upto 64 flows.

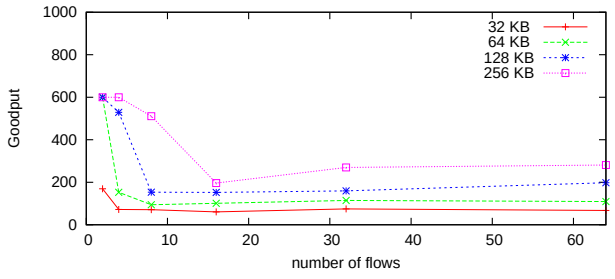


Fig. 14. Performance of Unmodified TCP in Mixed Protocol Setup

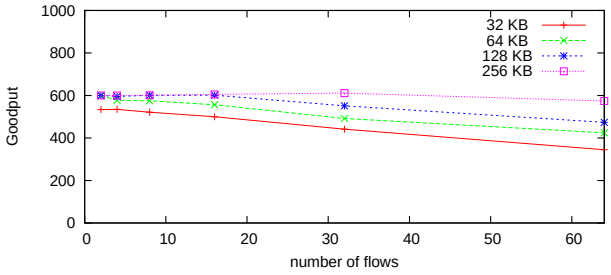


Fig. 15. Performance of Unmodified TCP in Mixed Protocol Setup

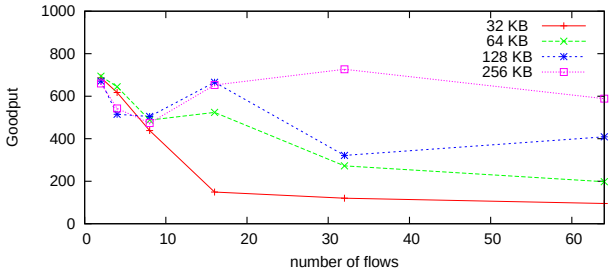


Fig. 16. Performance of Unmodified QCN in Mixed Protocol Setup

IX. Performance in QCN Baseline Simulations

Even though QCN with the performance modifications that we have proposed performs much better than unmodified QCN in a TCP Incast setup, we need to examine the effects this has on QCN performance in other workloads where QCN is intended to operate. The IEEE working group has a set of baseline simulations intended to be used for examining QCN performance in various setups. We have implemented four of the QCN baseline simulations in the ns-2 network simulator as faithfully as we could, with some minor differences. We have examined the performance of both QCN without modifications, as well as the modifications we have proposed.

We have avoided describing the simulation setups for the baseline simulations in the four baseline simulations for brevity, since they can be found in [19], [20], and [21] if needed. Upon running the baseline simulations for different versions of QCN, we noticed that: i. Congestion memory based sampling did not have much of an effect on QCN performance in the baseline simulations as compared to unmodified QCN performance. ii. R_AI modifications have only a slight effect on QCN performance: improving it somewhat, but not much. iii. Detailed sampling results are very similar to the results obtained from QCN based on [17]. iv. Even with combined modifications, the simulation results are dependent mostly only on the type of sampling used (as explained above for the two proposed modifications), with the RP side modifications not making much difference in either sampling method.

Overall, no negative effect was seen in the baseline simulations with the proposed QCN modifications in effect: both individually, and combined (sampling and R_AI based modifications).

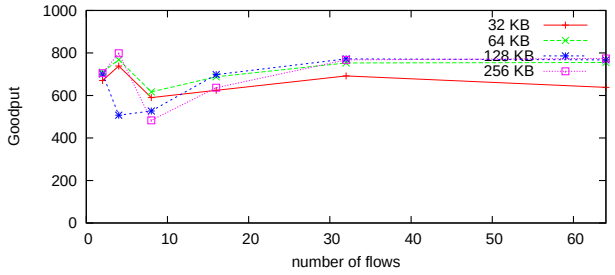


Fig. 17. Performance of QCN with Adaptive R_AI and Congestion Memory Based Sampling in Mixed Protocol Setup

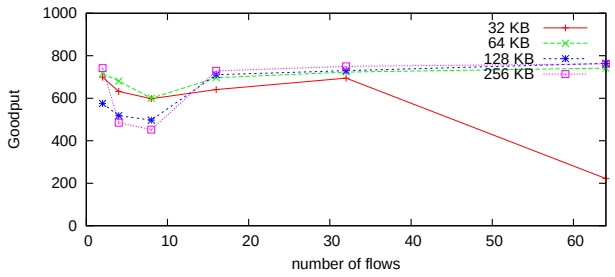


Fig. 18. Performance of QCN with Adaptive R_AI and Detailed Sampling in Mixed Protocol Setup

X. Conclusions

QCN is being investigated as a congestion control mechanism to be implemented in datacenters, where clustered storage and mixed protocol environments are both likely to be found. It is desirable to use switches with QCN to support various datacenter applications in order to support zero loss environments. If QCN does not work well in a clustered storage environment, this will be a barrier for its adaptation in datacenter environments. The ability of QCN to distribute bandwidth fairly not only among TCP flows, but also among flows that do not implement any rate control on their own, makes it a very desirable candidate for adaptation. We have examined the causes of QCN sub-optimal performance in a basic TCP Incast setup, and suggested several modifications that do not affect QCN behavior in normal setups, and are not very difficult/expensive to incorporate. Though these modifications do offer good link utilization, we believe it is possible to get better results yet, and this is an area which can be studied in more detail.

Acknowledgments

This work is supported in part by NSF grants 0702012 and 0621410 and by a grant from Qatar National Research Foundation. The authors would like to thank the Parallel Data Laboratory of Carnegie Mellon University, in particular Amar Phanishayee, for having made their ns-2 simulation code available, and Rong Pan and Dr. Balaji Prabhakar for offering suggestions during our implementation of QCN baseline simulations.

References

[1] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the Panasas parallel file system," in *Proc. USENIX Conference on File and Storage Technologies*, San Jose, CA, USA, 2008, pp. 17–33.

[2] Network file system version 4. http://www.ietf.org/dyn/wg/charter/nfsv4-charter.html. IETF. Last Accessed: March 16, 2010.

[3] Lustre File System. http://wiki.lustre.org/index.php/Main_Page. Sun Microsystems. Last Accessed: March 16, 2010.

[4] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates, "Quantifying performance and quality gains in distributed web search engines," in *Proc. 32nd international ACM SIGIR conference on research and development in information retrieval*. Boston, MA, USA: ACM, 2009, pp. 411–418.

[5] L. Peterson and B. Davie, *Computer Networks: a Systems Approach*. San Francisco, CA: Morgan Kaufmann Publications, 2007.

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *Online*, 1996, http://www.ietf.org/rfc/rfc1889.txt, Last Accessed: March 24, 2010.

[7] B. Ford, "Structured streams: a new transport abstraction," in *Proc. SIGCOMM '07: 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2007, pp. 361–372.

[8] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *Proc. FAST'08: 6th USENIX Conference on File and Storage Technologies*. San Jose, CA, USA: USENIX Association, 2008, pp. 1–14, http://portal.acm.org/citation.cfm?id=1364813.1364825.

[9] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. SIGCOMM09*, Barcelona, Spain, 2009, pp. 303–314.

[10] V. Vasudevan, H. Shah, A. Phanishayee, E. Krevat, D. G. Andersen, G. R. Ganger, and G. A. Gibson, "Solving TCP Incast in Cluster Storage Systems," *Fast2009*, 2009.

[11] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, C. Faster, and D. Maltz, "DCTCP: Efficient Packet Transport for the Commoditized Data Center," *Microsoft Research (online))*.

[12] 802.1Qau - Congestion notification. http://www.ieee802.org/1/pages/802.1au.html. Last Accessed: March 16, 2010.

[13] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using infiniband architecture congestion control," *Proceedings HP-IPC 2005*.

[14] Y. Birk and V. Zdornov, "Improving communication-phase completion times in HPC clusters through congestion mitigation," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, pp. 1–11.

[15] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, and G. A. Gibson, "A (In)Cast of thousands: scaling datacenter TCP to kiloservers and gigabits," *Pittsburgh, PA: CMU-PDL-09-101: Carnegie Mellon University Parallel Data Lab Technical Report*, 2009.

[16] R. Pan, "QCN pseudocode version 2.2," *IEEE EDCS-608482*, 2008, http://www.ieee802.org/1/files/public/docs2008/au-pan-qcn-serial-hai-2-1-0408.zip.

[17] ——, "QCN pseudocode released in 2007 November," *IEEE EDCS-608482*, 2007, http://www.ieee802.org/1/files/public/docs2007/au-rong-qcn-serial-hai-pseudo-code-0711.pdf.

[18] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. SIGCOMM '89: Symposium on Communications architectures & protocols*. New York, NY, USA: ACM, 1989, pp. 1–12.

[19] B. Atikoglu, A. Kabbani, R. Pan, and B. Prabhakar, "QCN: Second batch of benchmark simulations," IEEE, Tech. Rep., 2008, http://www.ieee802.org/1/files/public/docs2008/au-sim-rong-qcn-hai-0108.pdf.

[20] ——, "QCN: An update of benchmark simulations," IEEE, Tech. Rep., 2008, http://www.ieee802.org/1/files/public/docs2008/au-sim-rong-qcn-hai-updatesimu-0108-1.pdf.

[21] ——, "QCN: Benchmark simulations - scenario 4," IEEE, Tech. Rep., 2008, http://www.ieee802.org/1/files/public/docs2008/au-sim-rong-qcn-hai-0208.pdf.