# PERFORMANCE OF QUANTIZED CONGESTION NOTIFICATION IN TCP INCAST IN DATA CENTERS

A Thesis

by

PRAJJWAL PRASAD DEVKOTA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2010

Major Subject: Computer Engineering

# PERFORMANCE OF QUANTIZED CONGESTION NOTIFICATION IN TCP INCAST IN DATA CENTERS

A Thesis

by

PRAJJWAL PRASAD DEVKOTA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

| | |
|---|---|
| Chair of Committee, | A.L. Narasimha Reddy |
| Committee Members, | Srinivas Shakkottai |
| | Dmitri Loguinov |
| Head of Department, | Costas N. Georghiades |

May 2010

Major Subject: Computer Engineering

# ABSTRACT

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

(May 2010)

Prajjwal Prasad Devkota, B.E., Birla Institute of Technology, Mesra, India

Chair of Advisory Committee: Dr. A. L. Narasimha Reddy

This thesis analyzes the performance of Quantized Congestion Notification (QCN) during data access from clustered servers in data centers. The reasons why QCN does not perform adequately in these situations are examined and several modifications are proposed to the protocol to improve its performance in these scenarios. The causes of QCN performance degradation are traced to flow rate variability, and it is shown that adaptive sampling at the switch and adaptive self-increase of flow rates at the QCN rate limiter significantly enhance QCN performance in a TCP Incast setup. The performance of QCN is compared against TCP modifications in a heterogeneous environment, and it is shown that modifications to QCN yield better performance. Finally, the performance of QCN with the proposed modifications is compared with that of unmodified QCN in other workloads to show that the modifications do not negatively affect QCN performance in general.

# ACKNOWLEDGMENTS

I would like to thank Dr. Reddy for his continuous guidance, feedback and support. I 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. I would like to thank Manish Singh for his help on understanding several ns-2 architecture concepts and getting on with my code, as well as Kiran Kotla, Kapil Garg, and other members of my research group for their help.

# TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

# LIST OF FIGURES

FIGURE Page

FIGURE Page

# LIST OF TABLES

# CHAPTER I

# INTRODUCTION

## A. Background

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

---

The journal model is *IEEE Transactions on Automatic Control.*

applications.

## B. 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.

Packet buffers in switches are expensive, and recent work [8] also explores the possibility of using small buffers in equipment and still achieving near full link utilization. However, increasing the number of servers in a distributed setup is not always beneficial when dealing with low buffer switches. We introduce a very simple setup in Figure 1 (based on [9]), which we shall use both to explain Incast, and to study the performance of various protocols in a setup causing Incast later in this thesis.

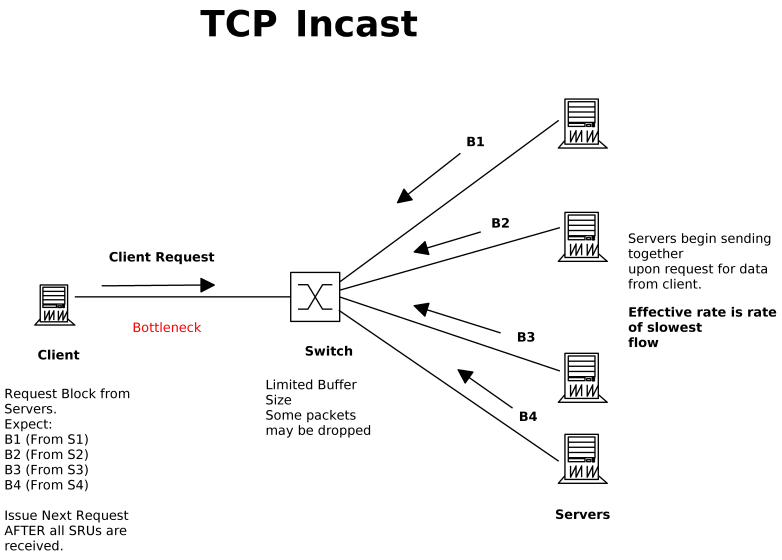



Fig. 1.: Basic Setup Subject to TCP Incast Collapse (Based on [9])

The setup in Figure 1 (based on [9]), consists of a client node, which requests data from multiple servers at once. An example scenario where this could happen is one where data is stored in fixed size blocks, which are striped into smaller chunks and stored across the servers. When the client needs to read data, it sends requests to all the servers which have the data stored. Each server sends back the stripe of data it has stored back to the client. This response from the server is known as a Server Response Unit (SRU). For purposes of simplicity, let us assume that the client is able to only request one block of data at a time. This implies the following sequence of events: i. client sends request to servers ii. servers send back SRUs to client iii. after receiving all SRUs, client issues another request.

### 1. Barrier Synchronized Requests and Minimum Effective Rate

In the setup in Figure 1, the client sends a request to all servers simultaneously, receives the SRUs from all servers, and sends its next request. This kind of a setup where the client does not send a new request till all responses to the current request have been received is known as a barrier synchronized setup.

Let us consider that there are 10 servers (hence flows) in the setup. This would imply that each flow should get an effective rate of 100Mbps and respond to the client. However, let us assume that for some reason, one flow is only able to transmit at 50Mbps. In a normal TCP setup, the other flows would take advantage of this extra 50Mbps bandwidth, and the link would still be fully utilized.

However, in a barrier synchronized setup, the other flows can only transmit the currently requested SRU, after which they have to wait for the slow (50Mbps) flow to complete its SRU transmission, after which the client issues a new request. This implies that the effective time for transmission of a SRU is the longest time taken among all the flows, and thus, that the effective rate of all flows is the minimum rate

among all flows. This effect is illustrated in Figure 2.

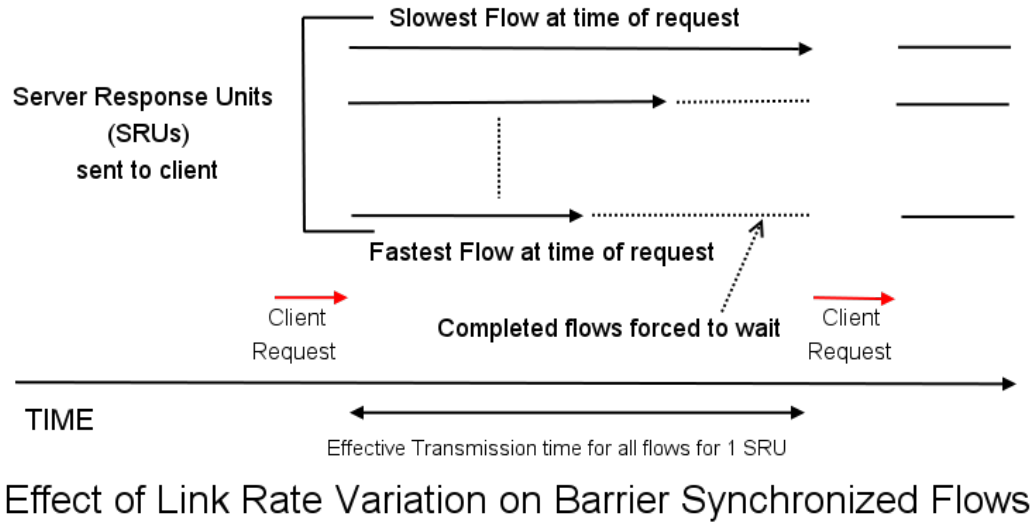



Fig. 2.: Effect of Link Rate Variation on Barrier Synchronized Flows

## 2. TCP Timeouts

As seen in Figure 1, the client, as well as each of the servers, are connected to a single switch. The physical capacity of each link is the same. In our simulations, we allocate 1Gbps to each link.

As the client sends a request to all servers at once, all servers send back their SRUs almost simultaneously back to the client. This leads to a bottleneck at the switch -> client link, where the input is n (number of servers) Gbps, while the output link is just 1Gbps. Congestion leads to packet drops, and packet drops in turn, tell TCP that there is congestion.

The congestion control mechanism of TCP quickly cuts rates down, and soon, all flows transmit at roughly C/n capacity, where C is the link capacity, and n is the number of servers. However, for the congestion control mechanism of TCP to work

properly, it must detect losses. This is done by two mechanisms [5] : i. TCP Fast Retransmit based on DUPACK detection and ii. Timeout based retransmit. TCP is able to detect packet drops through duplicate DUPACKs with a smaller number of flows, but as the number of flows goes up, TCP timeouts also begin occurring. We will examine the reasons for these timeouts occurring in coming sections, but for now, let us consider the effect a timeout has on a flow.

The TCP based timeout mechanism causes a flow to timeout if an acknowledgment is not received for a packet after 2x the estimated Round Trip Time (RTT) [5]. The Retransmit Timeout (RTO) value thus calculated however, has a lower limit of 200ms in current TCP implementations. A smaller lower bound was previously not necessary due to higher link latencies, and lower RTO settings also require a finer clock granularity. Let us assume that a SRU is of 256KB size, implying that the time to transmit the SRU with a 1Gbps link would be roughly 2ms. However, because of this timeout, the SRU transmission cannot be completed until the timeout expires, and the effective flow rate is decreased greatly.

In a setup where the TCP flows are not synchronized with each other, this does not pose much of a problem. However, as seen in subsection 1, the effective rate of all flows is equal to that of the timed out flow, leading to a drastic drop in link utilization.

## 3. Incast Problem in Literature

It has been seen that after timeouts begin to occur significantly, TCP performance drops drastically, and is known as the TCP Incast problem [9]. An illustration of the TCP Incast effect in our own simulations can be seen in Figure 3. As can be seen from the figure, link utilization drops drastically after a certain number of flows is exceeded, and the number of flows supported before this performance collapse occurs

is roughly doubled with a doubling in the switch buffer size. These results are based on the simulations done by Phanishayee et al. in [9].

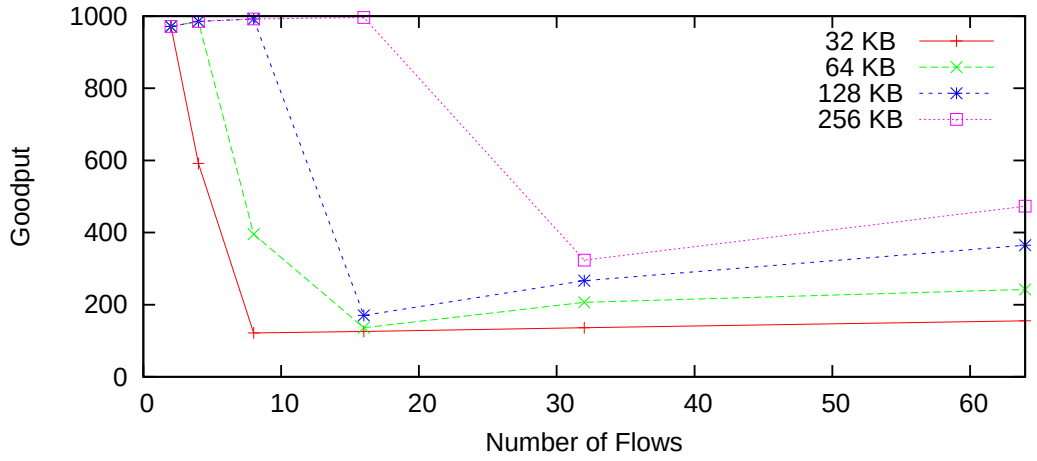



Fig. 3.: TCP Incast Collapse

The reason for this drop in network throughput is because of the interplay of heavy congestion (and hence packet drops), TCP rate control and timeout mechanisms, and the synchronized nature of the traffic.

A nice summary of the preconditions for TCP Incast is stated in [10], where the preconditions are listed as: High-bandwidth, low-latency networks, Clients issuing barrier-synchronized requests, and servers returning data that is not sufficient to significantly utilize the link for a long period of time by itself.

Heavy packet drops due to severe congestion cause the TCP timeout mechanism to kick in (as opposed to DUPACKs preventing timeouts from being necessary [9]). The onset of TCP timeouts, as discussed in subsection 2, can cause a catastrophic performance collapse, thus leading to the TCP Incast problem.

TCP timeouts can occur due to several reasons [9]:

- An entire window of data is lost, so that no feedback is received at the sender

(Full Window Loss)

- Last packet in a Server Response Unit (SRU) is lost: since no further packets are sent before the client issues another request, no DUPACKs occur, causing a timeout to be necessary (Last Packet Loss)
- Retransmitted packet (due to DUPACKs/timeout) is also dropped (Lost Retransmit)

In a non-synchronized flow scenario, even if a single flow times out, the other flows continue transmitting, while adjusting their rates. Eventually, they all converge to rates are more or less evenly divided among all flows. However, in a synchronized transmission scenario, most flows will have already finished transmitting their data before the victim flow(s) finish transmission after timeout. After the timeout period has elapsed, the affected flow will reduce its window and go into Slow Start, but the other flows will rapidly increase their rates until they detect packet drops, by which time another timeout will likely have occurred. The timeout mechanism, while effective in most setups, incurs a very heavy penalty in the setup in question, leading to continuously poor performance with the onset of TCP timeouts.

## 4. Silent Periods during TCP Incast Collapse

An interesting observation is the occurrence of 'silent periods' in the simulations. A 'silent period' is the period when a victim flow waits for a TCP timeout to occur, while other flows wait for the client to send another request. Very little traffic flows through the switch $->$ client link during these intervals. A table of total times during which no traffic was seen along the bottleneck link was generated for each set of simulations, and in each simulation, a very strong correlation was observed between the onset of long periods spent in silence with the onset of Incast. The table

of total seconds spent in silence (over a period of 20 seconds) is shown in Table I for unmodified newreno:

Table I.: Silent Period in Seconds for Unmodified TCP Simulations

| **Flows** | **32** | **64** | **128** | **256** |
|---|---|---|---|---|
| **2** | 0 | 0 | 0 | 0 |
| **4** | 6 | 0 | 0 | 0 |
| **8** | 16 | 9 | 0 | 0 |
| **16** | 16 | 15 | 15 | 0 |
| **32** | 15 | 14 | 14 | 12 |
| **64** | 14 | 12 | 12 | 12 |

As can be seen from the table, after Incast collapse occurs, the amount of time during which no data is sent increases significantly. The actual observed link utilization is even less than the 'active time'/total simulation time, presumably because of the way the silent periods are calculated: only periods with no traffic are counted, and if a period has minimal traffic, it is not counted as a silent period. For example, for 4 flows and 32 KB buffer size, 6 seconds (out of 20) are spent in total silence. The actual observed goodput for this simulation is 647 Mbps (64.7% utilization), while 14/20 gives us 70% utilization.

## C. Current Approaches to Dealing with Incast

Increasing the SRU size and increasing the buffer size in the switch are two methods by which the onset of Incast can be pushed further away. The authors state that the number of servers supported before Incast collapse roughly doubles with a doubling

of the buffer size in [9]. Ethernet flow control [11] is also effective for the simple setup of Figure 1, but is seen to fail in more complex setups because of problems with head of the line blocking [9].

The most effective method proposed in the literature so far deals with not trying to prevent TCP timeouts, but with simply reducing the penalty of the timeouts on the flows. Phanishayee et. al. propose 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 effect TCP performance in general scenarios. This modification reduces the penalty of TCP timeouts to a value that does not reduce the effective flow rates significantly. In our simulations using TCP with reduced minRTO, we noted that the onset of timeouts occur with roughly the same number of flows as in unmodified TCP, but a much greater number of actual timeouts occur while throughput is unaffected, which satisfies the aim of the TCP modification. However, no 'silent periods' are observed for the simulations, which is quite obvious, considering the fact that timeout penalties are considerably reduced.

However, while modified TCP is an effective solution to the basic TCP Incast problem, two questions still remain: i. the performance of TCP flows in a mixed protocol environment, and ii. the effect of introducing switches which have Quantized Congestion Notification, which is being standardized by the IEEE 802.1 QAU group [12] and will likely be implemented in data center switches. While TCP might perform well in a homogeneous environment, a data center environment consist of various other protocols such as UDP, which are much more aggressive. This mix of protocols can have a detrimental effect on TCP performance, and the question of how TCP can be made to perform better in such a setup arises.

In the following chapters in this thesis, we will talk about Quantized Congestion Notification (QCN), and the performance of TCP flows in an environment where QCN

is enabled in the switch. We will analyze the causes of poor performance of Incast flows in a setup with QCN enabled, and propose various modifications to alleviate these problems. Further, we show that these modifications do not have a detrimental effect on QCN performance in other workloads.

# CHAPTER II

# 802.1QAU - CONGESTION NOTIFICATION

## A. Introduction

The full title of the IEEE 802.1 QAU Congestion Notification PAR is IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification. This standard specifies protocols, procedures, and objects for congestion management of long-lived data flows in low bandwidth delay product networks [12]. The switches employed are able to send congestion information to traffic sources, which are able to rate limit their flows to avoid frame loss, regardless of the transport level protocol being used. The standard recently adopted by the committee is Quantized Congestion Notification (QCN). Two other protocols that had previously been examined by the commitee along with QCN were Ethernet Congestion Manager (ECM, previously known as Backward Congestion Notification or BCN) [13], and Forward Explicit Congestion Notification (FECN) [14].

## B. Quantized Congestion Control

### 1. Overview

QCN consists of two main components: the Congestion Point (CP) and the Reaction Point (RP). An illustration of QCN components, including the traffic source and destination, is shown in Figure 4 (based on [15]). The CP monitors for congestion, and notifies the RP when congestion occurs, including the flow id of the offending flow.

When transmitting data from a traffic source to its destination host, all interme-

diate switches can become congested, and are hence called congestion points. These switches (CPs) regularly sample packets and monitor their queue lengths to make sure congestion is not occurring, based on the growth rate of the queue and its offset from an equilibrium 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 6 bit value (between 0 and 63), leading to the name Quantized Congestion Notification.

The RP has rate limiters for various flows, and can adjust the rate of each offending flow individually. Based on feedback from the CP, the RP adjusts the rate of the flow. Negative feedback causes multiplicative rate decrease for the offending flow. Rate increase is more complex, and has three different behaviors, depending on how when the last negative feedback message was received. There is no positive feedback as such in QCN, only the lack of negative feedback. These stages will be described in more detail later.

### 2. Congestion Point

To control congestion, each Congestion Point (intermediate switch) samples forwarded packets with a frequency that depends upon the amount of congestion encountered. During heavy congestion, packets are sampled more frequently, while packets are sampled less often is there is less congestion. Congestion is determined based on the feedback value calculated. The feedback value thus calculated, if negative, is put in a feedback message, along with the flow identification information (flowid), and sent back to the RP, which takes appropriate action based on the feedback.

Feedback is calculated on the basis of offset (qoff) from an equilibrium queue length (QEQ) and the rate of growth of the queue (qdelta). It can be expressed as:

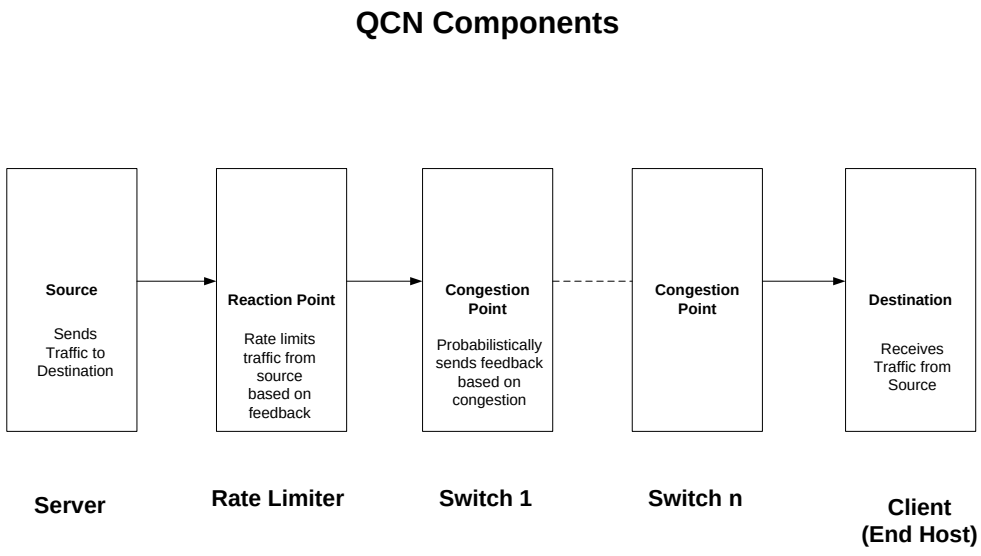



Fig. 4.: QCN Components (Based on [15])

$$qdelta = qlen - qold$$

$$qoff = qlen - QEQ$$

$$Fb = -[qoff + W * qdelta]$$

where W is a weight parameter, set to 2 in our simulations. The calculated feedback is quantized to 6 bits (values between 0 and 63), and if the feedback is negative, then a feedback message is sent back to the source.

### 3. Reaction Point

The RP is the part of the QCN mechanism that does the actual rate limiting of flows based on feedback from the CPs. A single RP can have multiple rate limiters, one for each offending flow. These rate limiters use rate limiting mechanisms such as Token Bucket or Leaky Bucket [5] to physically rate limit each offending flow individually.

An overview of the QCN rate control mechanism is given Figure 5.

When a Reaction Point (RP) receives a feedback message, it performs multiplicative decrease and goes into Fast Recovery mode. The rate at the time negative feedback was received is set as the target rate (TR), and the current rate (CR) is reduced according to the equation:

$$CR = CR * (1 - Gd * |Fb|)$$

During Fast Recovery mode, the new current rate is set to the average of the current rate and the target rate, while the target rate is left unchanged. Thus, the current rate increases very rapidly the first time congestion is indicated, then it gradually converges towards the rate at which negative feedback was received. The CP however, does not send any positive feedback. RPs perform self-increase periodically

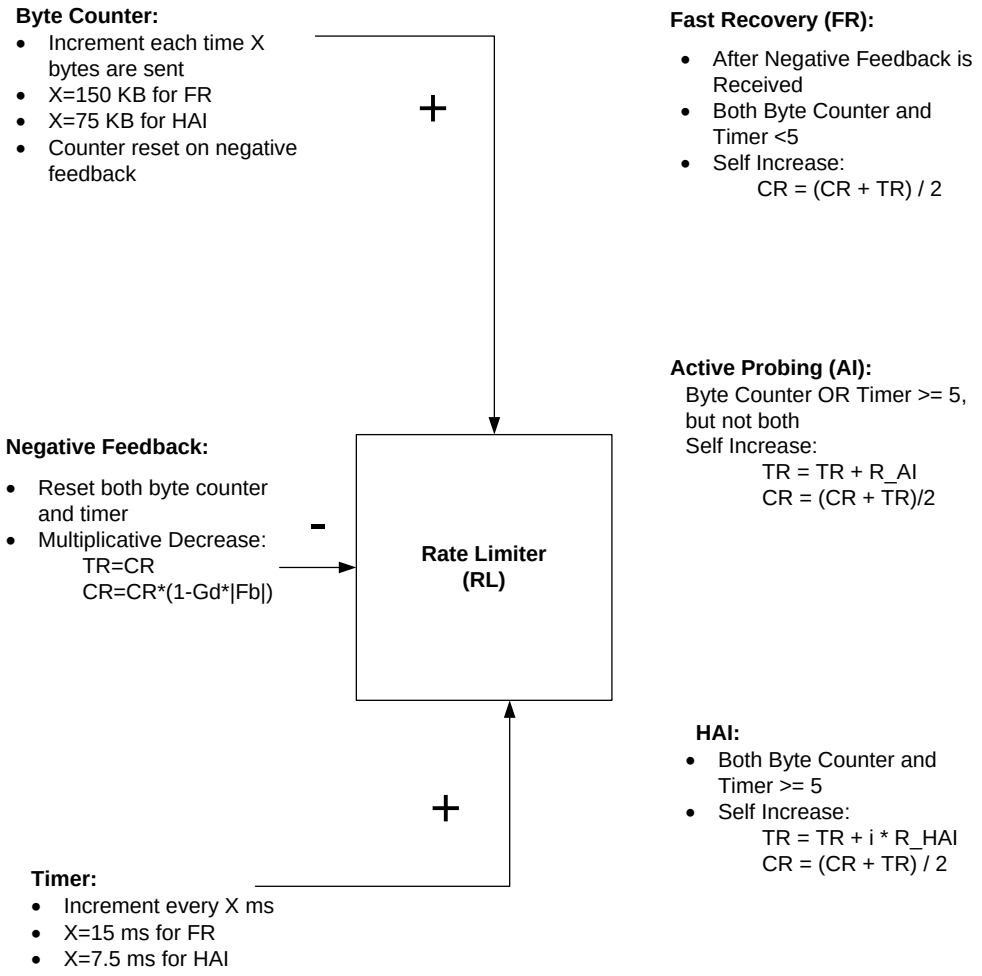



Fig. 5.: QCN Overview (Based on [16])

on the basis of lack of negative feedback using the two mechanisms below:

- **Time elapsed**: Each RP has a timer that is activated when negative feedback is received. It also has a counter than keeps track of the number of timer cycles elapsed since the last negative feedback was received. We shall refer to this as the timer state. The RP timer expires every 15ms during the Fast Recovery stage and Active Probing stage. During the HAI phase, the timer expires every 7.5ms. The timer state is incremented by 1 every time the timer expires, and self increase is also performed.

- **Bytes transmitted**: The RP also has a byte counter which keeps track of bytes transferred. The byte counter is incremented each time 150KB is transferred during Fast Recovery and Active Probing,while it is incremented each 75KB during HAI stage. Similar to incrementing the timer state, an increment of the byte counter also signifies self increase.

Self increase as referred to above implies the increase of the RP target rate according to the phase QCN is currently in. The policy of rate increase differs depending upon whether QCN is in the FR, AI, or HAI stage. The working of the three phases is described below:

- **Fast Recovery (FR)**: QCN is in FR after negative feedback is received, and as long as both the timer state and byte counter are below 5. During this stage, current rate (CR) is incremented with each self increase according to the following equation:

$$CR = (CR + TR)/2$$

- **Active Probing (AI)**: During this stage, QCN is the least aggressive. During each self increase, the Target Rate is increased by R_AI, and CR is increased

as before:

$$TR = TR + R\_AI$$

$$CR = (CR + TR)/2$$

- **HAI**: This is a stage of aggressive probing, which occurs when both the byte counter and timer state are greater than or equal to 5. QCN increases the target rate by R_HAI multiplied by the number of times the event (timer elapse or required number of bytes transmitted) has elapsed. It can be written as:

$$TR = TR + i * R\_HAI$$

$$CR = (CR + TR)/2$$

where i is the event count. Figure 5 (based on [16]) shows a summary of the workings of QCN, while Figure 6 (based on [15]) shows rate changes during FR and AI stages.

## 4. Comparison of QCN and TCP Rate Control Mechanisms

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 allows applications with different transport protocols (TCP and UDP) to fairly share bandwidth 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, as a function of the available physical bandwidth. It is hard limited by the rate assigned to it by the QCN RP. An approach that modifies TCP to reduce

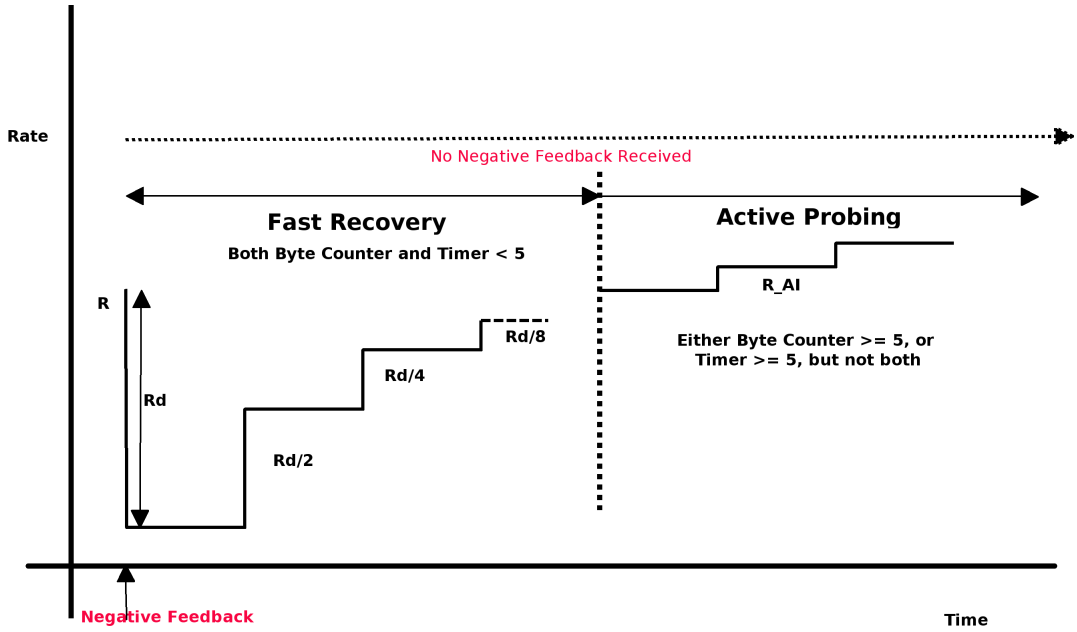



Fig. 6.: QCN Fast Recovery and Active Probing (Based on [15])

minrto aims not to prevent timeouts, but to remove the penalty incurred by the onset of timeouts. In our Incast 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* as appropriate 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 data center 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 this thesis the effectiveness of QCN in TCP Incast scenarios is studied. Simulation results show that QCN as per current specifications is not very

effective in controlling the congestion collapse in the Incast scenarios. The causes of this are analyzed, and two modifications to QCN to improve its performance in TCP Incast scenarios are proposed. It is shown that QCN with the proposed enhancements can be an effective solution in mixed workloads with TCP and UDP in data centers, while continuing to function effectively in other workloads as well.

# CHAPTER III

# PERFORMANCE OF UNMODIFIED CONGESTION NOTIFICATION PROTOCOLS IN THE INCAST SCENARIO

## A. Evaluation of QCN in TCP Incast Scenario

In Chapter I section B, the basic setup used for simulations was described. The setup will be explained in more detail below, and the performance of QCN in the TCP Incast scenario will then be presented.

Figure 7 shows the basic simulation setup, similar to what is used in [10]. 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.

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,...).

### 1. 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

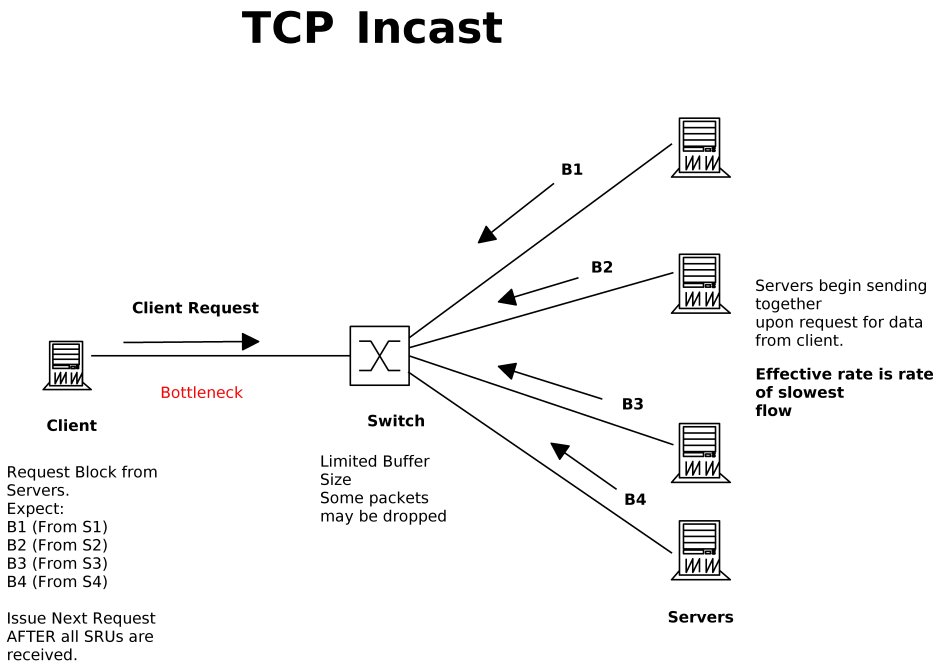



Fig. 7.: Simulation Setup for TCP Incast Collapse

for 256KB switch buffer size. The performance of unmodified newreno is shown in Figure 8. This was found to be in accordance with the results presented in [17].

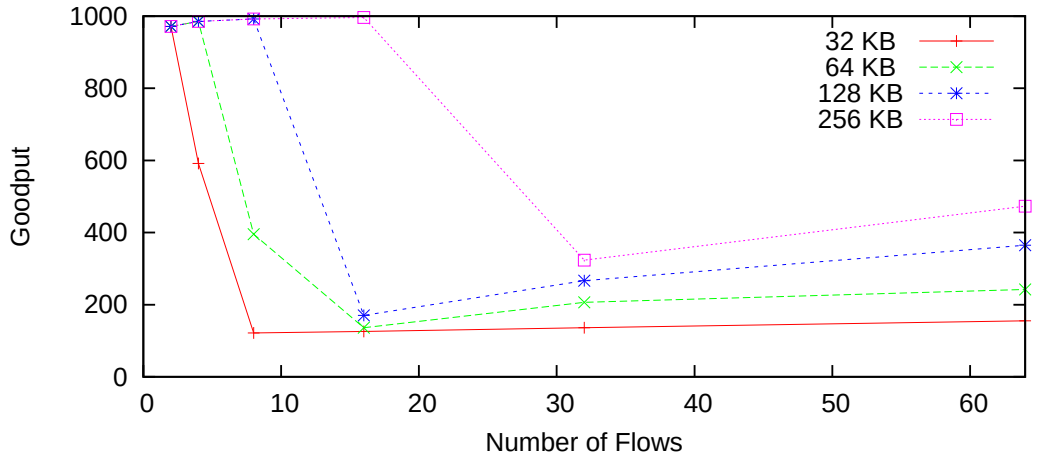



Fig. 8.: Unmodified TCP Newreno Goodput vs Number of Flows

### 2. Newreno with Reduced Minrto

We reduced the minimum retransmission timeout for TCP, and ran simulations with rtos of 10ms and 1ms, as suggested in [17]. The performance of TCP with a minrto changed to 10ms is shown in Figure 9. As seen in Figure 9, performance is much better, and there is almost 100% link utilization for 256KB buffer sizes, while goodput drops to 60% for 32 and 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 [17].

## B. QCN Performance in TCP Incast

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

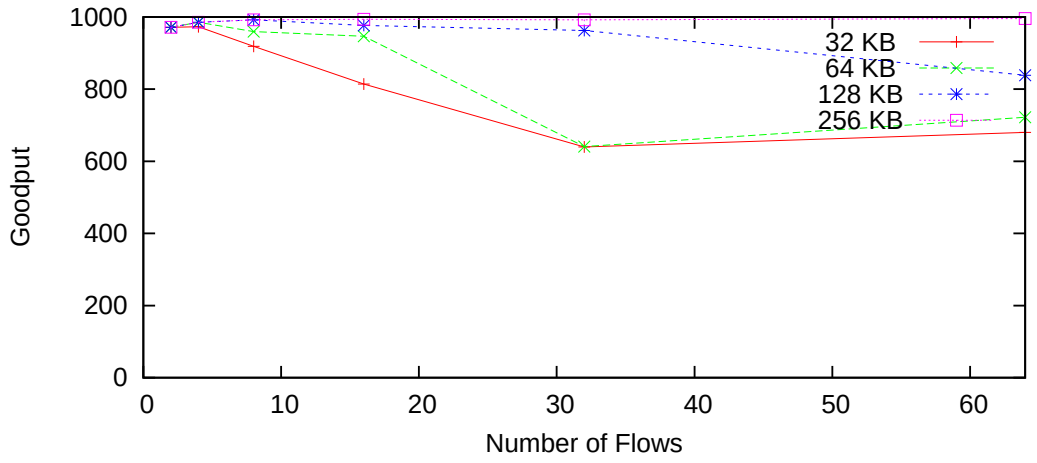



Fig. 9.: TCP with 10ms Minrto Goodput vs Number of Flows

Pan [18] [1]. We shall be referring to our implementation of QCN based on the latest available release of the pseudocode ([18]) as unmodified QCN, and all proposed modifications are applied to this implementation.

QCN can effectively control link rates very rapidly in a data center environment. However, it performs poorly in the TCP Incast setup in Figure 7. The performance of TCP Incast with QCN is shown in Figure 10.

Before analyzing QCN behavior, it is necessary to make a distinction between two regions in which QCN can operate, based on the number of flows (in the TCP Incast setup examined).

1. Before the onset of TCP timeouts (and hence 'periods of silence')

2. After the onset of TCP timeouts (and hence 'periods of silence')

QCN initially behaves in the first region, effectively rate limiting TCP flows, and

[1]This version of pseudocode employs slightly different sampling as compared to the previously released version [19]. The effects of this in both Incast and baselines are quite noticeable though, as are discussed in the appropriate sections.

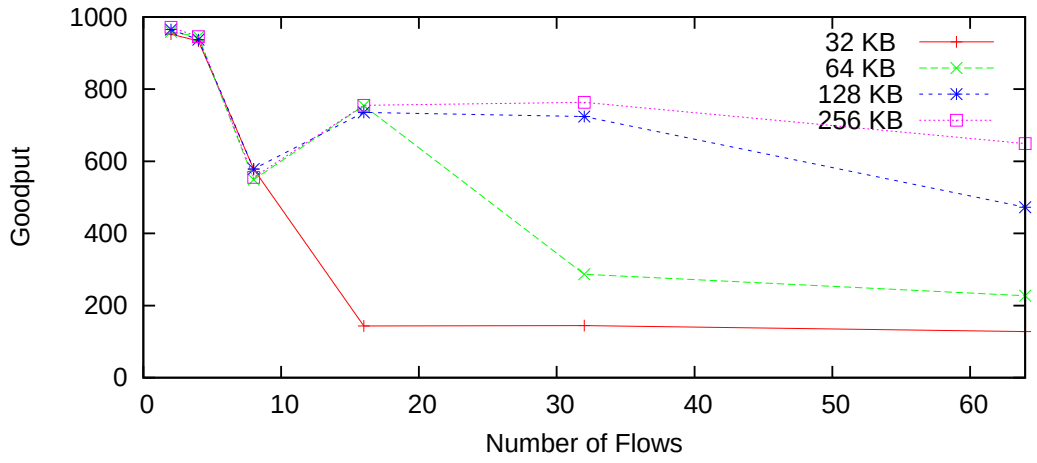



Fig. 10.: Unmodified QCN Goodput vs Number of Flows

preventing TCP timeouts from occurring as such. However, fairness issues play a big role in a barrier synchronized setup such as this one, causing variations in QCN flow rates to affect overall performance.

After TCP timeouts begin to occur, QCN flow rates fluctuate to a greatly, and behavior very similar to that of only TCP after Incast collapse is observed. These regions are examined in detail in the following sections.

### 1. QCN Behavior before Onset of TCP Timeouts

At first glance, it appears that the throughput collapses at 8 flows regardless of the buffer size. The throughput however, picks up for 256KB and 128KB after the initial drop at 8 flows. For 256KB buffer size, the throughput goes up to around 700Mbps, and goes only slightly lower for a higher number of flows. This is different from TCP behavior without QCN.

If the number of timeouts, which were seen to be the main cause of TCP Incast collapse, are observed, with QCN, the onset of timeouts is actually delayed. Timeouts

in normal newreno begin occurring at 4 flows, and at 16 flows, timeouts occur even for 256KB buffer size. On the other hand, timeouts begin significantly at 4 flows for 32KB, 8 flows for 64, 16 flows for 128KB, and 32 flows for 256KB buffer sizes with QCN enabled. The occurrence of TCP timeouts is shown in Figures 11 and 12.

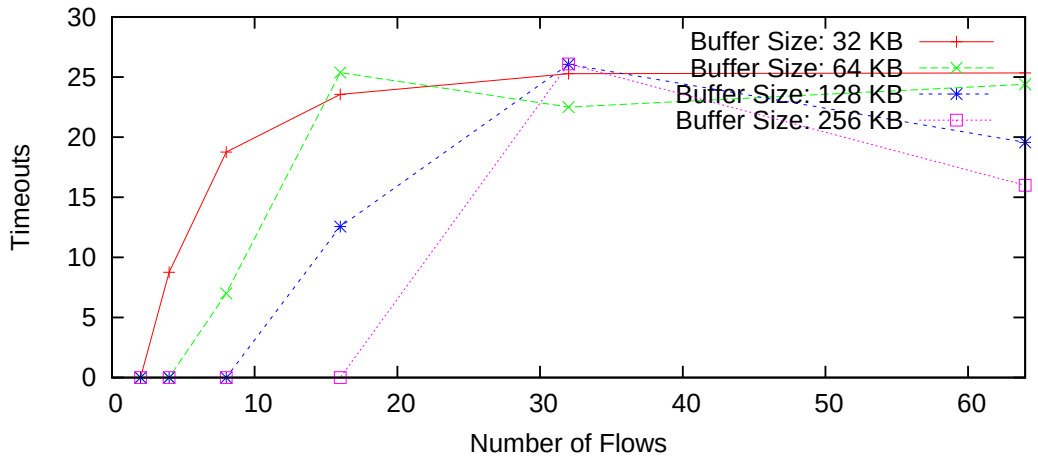



Fig. 11.: Timeouts per Flow in Unmodified TCP vs Number of Flows

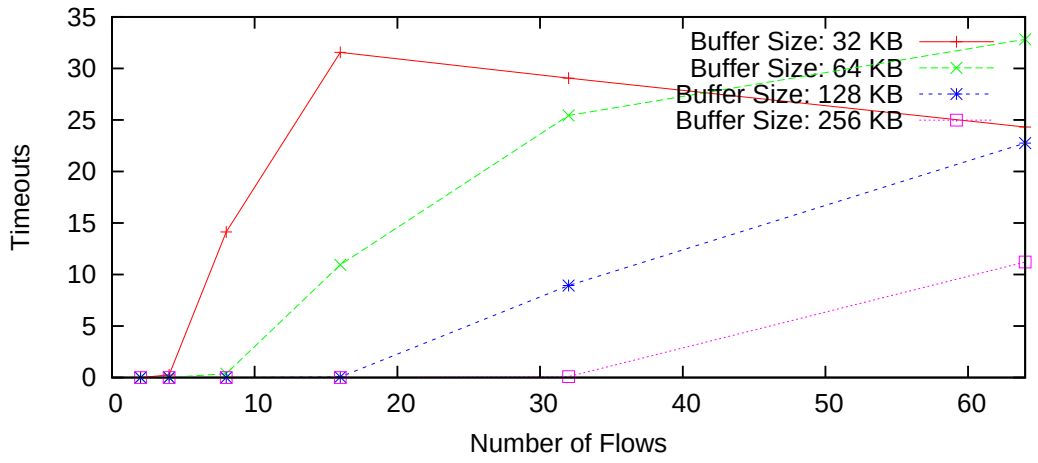



Fig. 12.: Timeouts per Flow in Unmodified QCN vs Number of Flows

The number of drops that occur before timeouts begin occurring is also much less

for newreno than for QCN, as shown in Figures 13 and 14. We note that before TCP timeouts begin to occur, the total number of queue drops is less than 1000. This would imply that TCP should be able to perform quite well with QCN enabled, and should achieve full link utilization. This is in fact, corroborated by observing the Congestion Window values with QCN enabled as compared to those with only newreno. If we compare the performance of newreno and QCN at 16 flows, with 256KB buffer size for example, the congestion window for the QCN enabled simulation goes up steadily to 450, while the average congestion window for unmodified newreno stays at around 30. At the end of this simulation (20 seconds), there are 35000 drops for unmodified newreno only, while there are only 40 drops for newreno on top of QCN.

With the data just quoted for the simulation with 256KB buffer size and 16 flows, the performance of QCN would be expected to be at least as good as that of unmodified newreno. However, while the goodput of newreno is constantly at full link utilization (1Gbps), the goodput achieved with QCN is at 700Mbps.

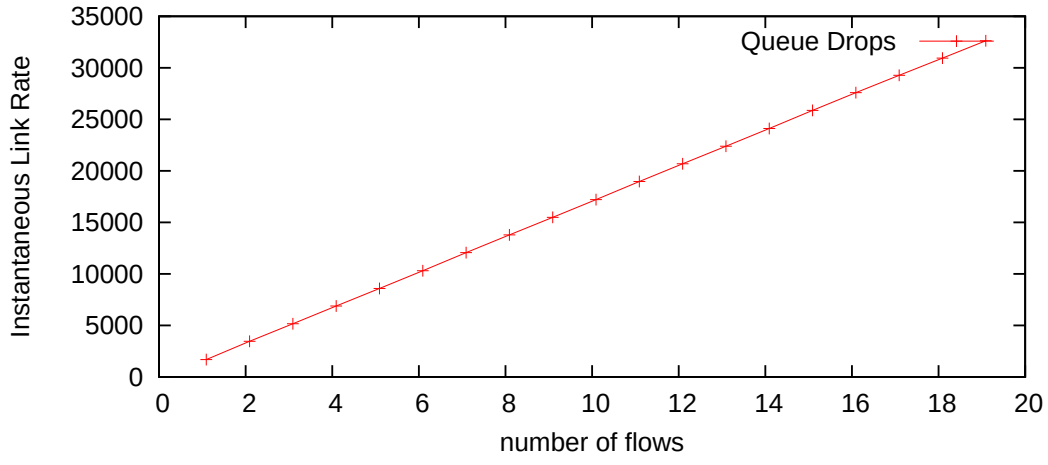



Fig. 13.: Unmodified TCP Drops with Time

All of the data above seems to indicate that though QCN performance drops

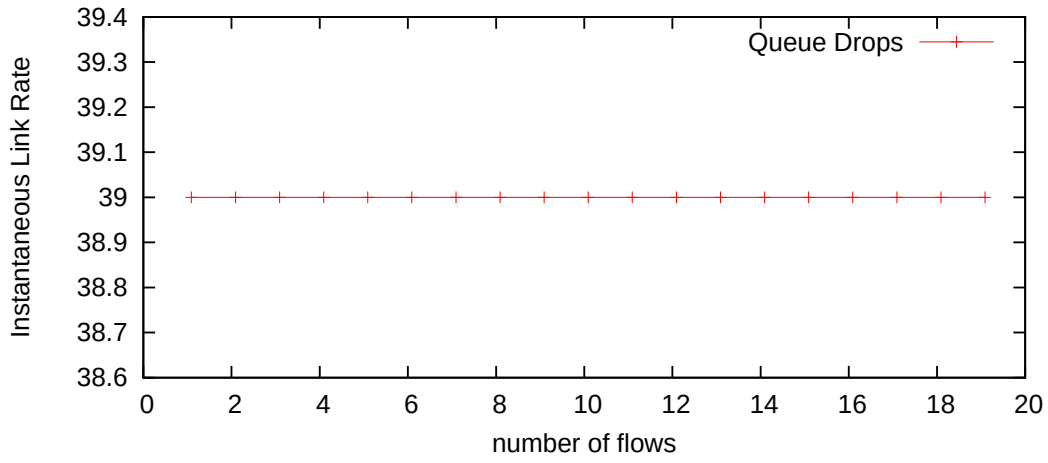



Fig. 14.: Unmodified QCN Drops with Time

after TCP timeouts begin to occur, its performance is also not very good before TCP timeouts occur. The onset of TCP Incast due to timeout penalties can be corroborated with the total silent time during a simulation, as shown in Chapter I section B subsection 4. As can be seen from Table II however, even when not much time is spent in a silent period (e.g. 256 KB buffer, 8 flows), the total link utilization is very low.

The reason for this poor performance of QCN cannot be explained by looking at TCP behavior alone. In fact, in Chapter I section B subsection 2, we discussed how TCP timeouts cause TCP Incast behavior, and without timeouts, TCP itself should perform without problems, as seen by the large congestion window size and small number of drops.

However, as explained in Chapter II section B subsection 4, rate limiting is also done by QCN, and if QCN allocates a certain rate to a flow, that allocation is equivalent to a physical limit on the bandwidth that can be utilized. Please refer to the example in Chapter I section B subsection 1, where one flow with an effective

Table II.: Silent Period in Seconds for Unmodified QCN Simulations

| **Flows** | **32** | **64** | **128** | **256** |
|---|---|---|---|---|
| **2** | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 |
| **8** | 13 | 0 | 0 | 0 |
| **16** | 13 | 10 | 0 | 0 |
| **32** | 14 | 8 | 5 | 0 |
| **64** | 15 | 10 | 5 | 1 |

bandwidth of 50Mbps slowed down all other flows. The poor performance of QCN can be explained when these two concepts are kept in mind: QCN allocates rates that are fair over time, but at small time scales, this allocation is not perfectly fair. There is a certain variation among allocated flow rates at any given time, and the minimum rate for a flow becomes the effective rate for all flows, leading to a noticeable decrease in total link utilization. This effect is examined in more detail later.

We make several observations from the results in Figure 10 immediately: i. QCN does not achieve full link utilization even when only unmodified TCP does and ii. 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.

First, let us consider QCN performance without TCP timeouts, 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 17 we see that past 8 flows, even with a modified 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 15, where we see that the minimum link rate is constantly below 80 Mbps throughout the simulation. However, as seen in Figure 16, 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, implying that there is not one particular victim flow, and over a longer time, rate allocation is fair.

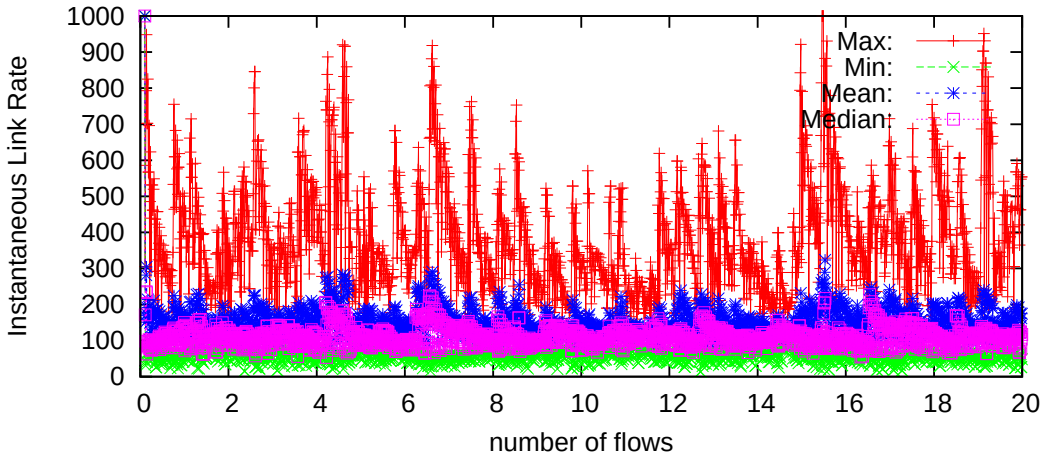



Fig. 15.: Link Rate Statistics for QCN 8 Flows 128KB Bufsize

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

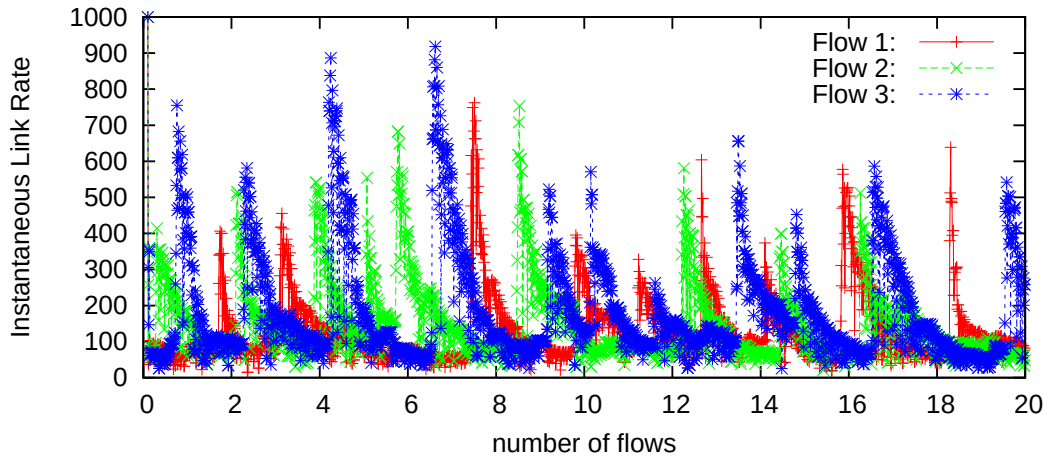



Fig. 16.: Link Rates for 3 Flows for QCN 8 Flows 128KB Bufsize

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*80=640Mbps.

Link rates have been sampled periodically in the simulation, and the minimum rate among all of the link rates is termed the minimum link rate for that instant. 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 * the number of flows should give the effective goodput for the simulation before TCP collapse occurs. This is corroborated by the data in Table III. 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 III.: Average Minrate and Goodput for Our Implementation of QCN for: 128 KB Buffer Size

| **Flows** | **Avg Minrate** | **Avg Minrate*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** | 29.40 | **940.77** | **325.01** | 8.94 |
| **64** | 32.03 | **2049.80** | **408.40** | 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 * flows) of all the flows. It is observed at this point, QCN is unable to control the congestion and timeouts start occurring and the timeouts increase as the number of flows are increased.

a. Performance of TCP with Reduced Minrto over QCN

According to the explanation of the bad performance of QCN given above, we can conclude that even when TCP does not have long intervals without traffic, link utilization is still low. This would imply that even with the TCP modifications proposed in [10], if QCN was also enabled in the setup, then the link utilization should be very similar to that of unmodified TCP performing over QCN, at least before TCP timeouts begin to occur significantly. This is confirmed by simulation: Figure 17 shows the performance of TCP with 10 ms minrto over a QCN enabled setup.

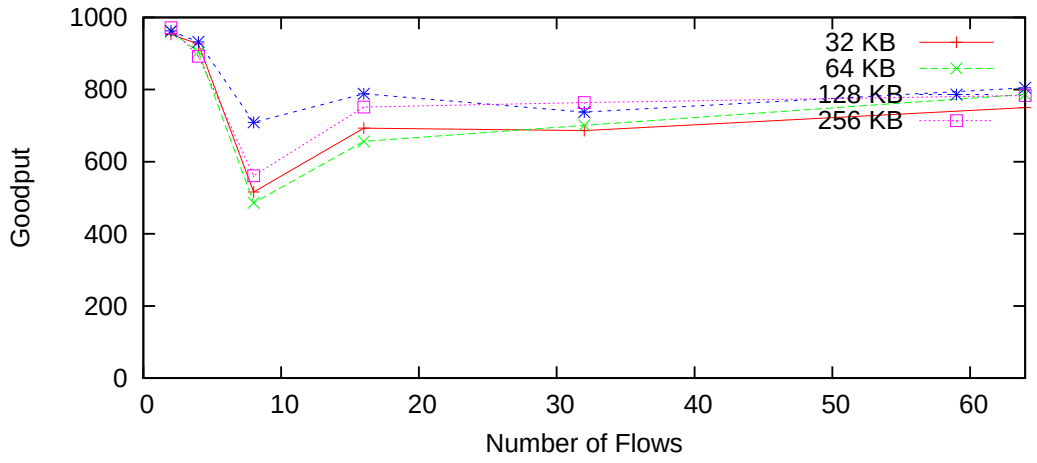



Fig. 17.: 10ms Minrto TCP with Unmodified QCN Goodput vs Number of Flows

It can be seen from Figures 10 and 17 that the performance of TCP with and without modified minrto is very similar with QCN enabled before TCP timeouts begin occurring.

### 2. QCN Behavior after Onset of TCP Timeouts

It can be seen from Table III that when TCP timeouts begin occurring (last column corresponds to TCP timeouts per flow), average minrate * 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. The actual traffic generated by the timed out flow is hence very low, and this low rate affects all other flows as they are all synchronized, leading to a very low utilization throughout. This effect is was discussed in Chapter I section B subsection 1.

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 5Mb). 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 13*5=65MB rate increase could occur in the link rate (if in the AI region of QCN). 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 III, where the avg. min rate* num flows exceeds the link capacity of 1 Gbps for 64 flows.

An illustration of increases in QCN rate due to no negative feedback is shown in Figures 18 and 19. Note that flow 1 steadily has more rate allocated to it till around 5.6 seconds, when there is a sharp drop in rate. This corresponds to a spike in the

instantaneous throughput of flow 1. Flow 2 is also seen to have become active at the same time, and other flows (not shown in figure) are also seen to become active. Further, there is a drop in the rate allocated to flow 2 as well at around the same time. After this, both flow 1 and flow 2 do not increase their rates during the period the flows are transmitting ( 5.7s for flow 1, nearly 5.8 seconds for flow 2). Flow 3 is apparently under timeout at this period, and is not transmitting. Its allocated rate is steadily increasing, and soon after it begins transmission at around 5.84 seconds, a corresponding drop in its allocated flow rate is seen.

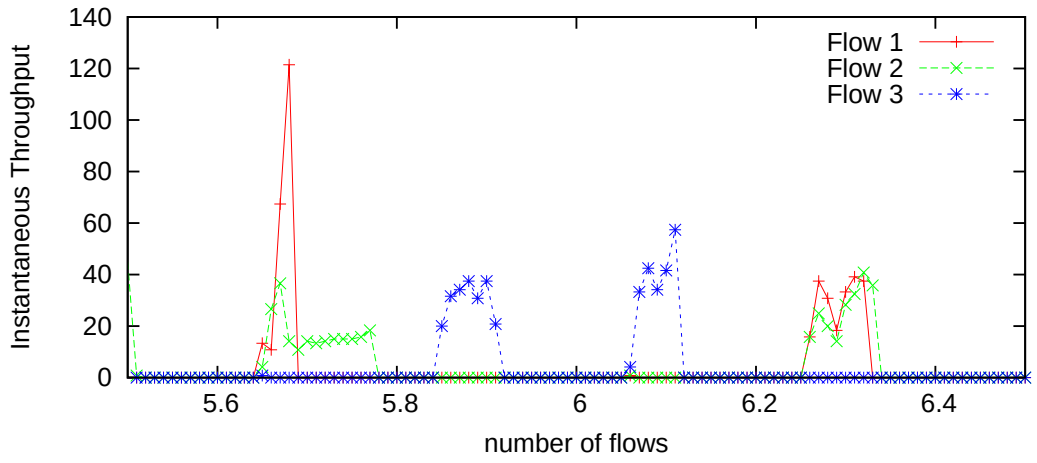



Fig. 18.: Instantaneous Throughput for 3 Flows from 5.5s-6.5s: 128KB Bufsize, 64 Flows

### 3. Perfect Rate Distribution

In subsection 1, it was pointed out that the poor performance of QCN was because of small timescale rate variations in a barrier synchronized setup, where all flows would effectively have an effective rate equal to the minimum rate among all flows. A natural question that arises is: if QCN were able to perfectly control rates among

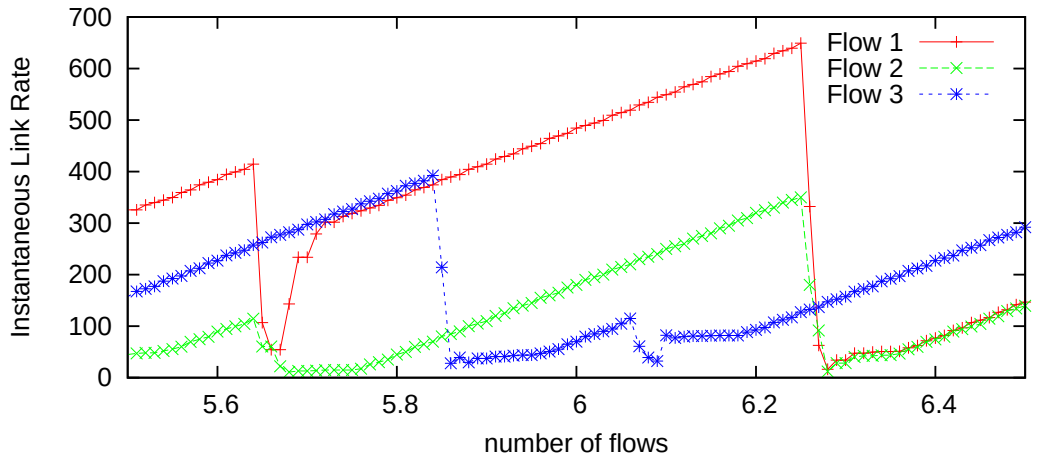



Fig. 19.: QCN Flow Rate for 3 Flows from 5.5s-6.5s: 128KB Bufsize, 64 Flows

flows, so that if there are n flows competing for a link of capacity C, then each flow would be allocated a rate of C/n, would QCN performance in a TCP Incast setup improve? In a sense, this would be the ideal case for QCN.

The simulation for this scenario is quite simple to implement: each flow is allocated a physical capacity equal to C/n, where n is the total number of flows. The same simulation set was run, and the results are shown in Figure 20. It can be seen from the figure that all flows, irrespective of buffer size, have almost complete link utilization. Further, though the figures have not been included for brevity, there are no queue drops or TCP timeouts for any number of flows and any buffer sizes, and the average queue length is less than 1 for all simulations.

These results strongly suggest that if QCN is able to achieve perfect rate distribution, its performance in a TCP Incast setup should considerably improve.

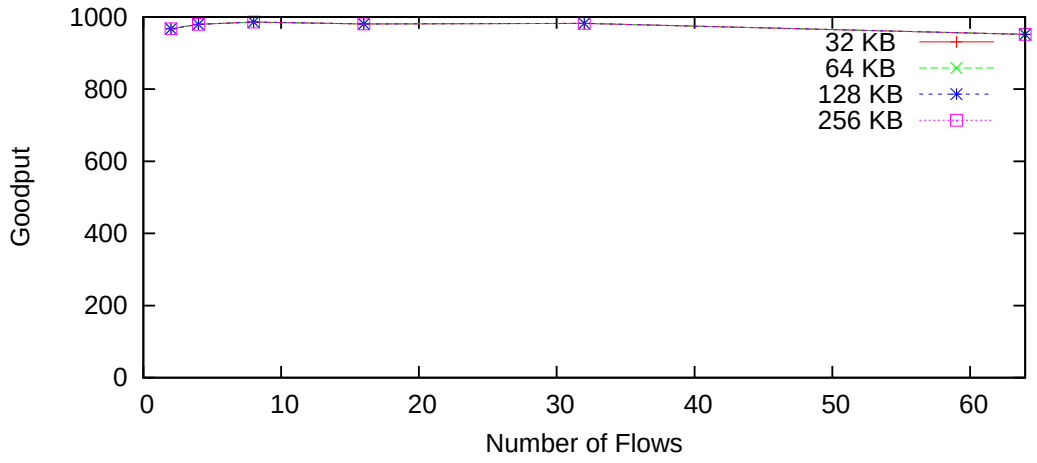



Fig. 20.: C/n Rate Distribution Goodput vs Number of Flows

## 4. Fair Queuing

While it is not clear on how a perfect rate distribution can be achieved, achieving better fairness at small timescales is desirable. Fair Queuing [5] is useful for providing fairness at small timescales. Simulations were run with both Stochastic Fair Queuing (SFQ) [20] and Deficit Round Robin (DRR) [21] to see if performance would improve over newreno performance, with an intent of using fair queuing on top of QCN if results showed significant improvements. These results are shown in Figures 21 and 22.

The results of SFQ at the switches are not seen to be much better than than of unmodified TCP, while those of DRR are somewhat improved for 32 and 64 KB buffer sizes before collapse, while results are not much better for higher buffer sizes. A simulation using SFQ with queue allocations hard limited, where each flow was assigned Q/n queue length, where Q is the total queue capacity and n is the number of flows was also carried out. The results for this simulation are shown in Figure 23.

It was found that in order to get SFQ to support upto 64 flows without Incast

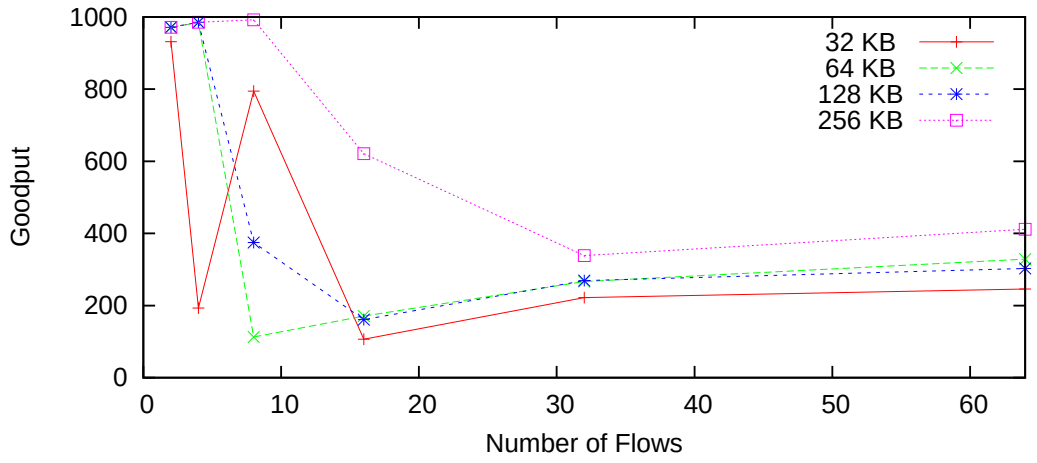



Fig. 21.: Unmodified TCP with SFQ at Switches Goodput vs Number of Flows

collapse, 1 MB buffer was required. Similarly, for DRR, 2 MB buffer was required, and for simple droptail, 2 MB buffer was required. SFQ with hard partitioning required 2 MB buffer size as well. Though Fair Queuing provides fairness among flows, it was seen not to be very effective in the Incast setup with small buffer sizes.

As none of the simulation setups yielded much performance benefit over simple droptail queuing at the switches, the implementation of QCN with these mechanisms was not carried out.

## 5. Improvements to QCN

Two major factors were seen to have an effect on QCN performance: before and after TCP timeouts begin occurring:

1. High variation among QCN flow rates at small timescales
2. Rapid increase of QCN flow rates during 'silent periods' due to TCP timeouts

Modifications to QCN that would help improve its performance in the Incast setup need to address these issues effectively.

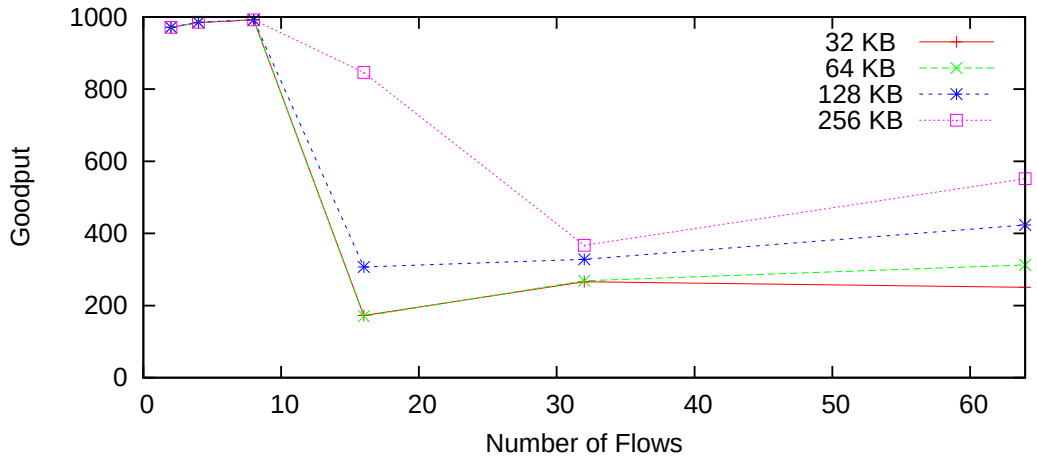



Fig. 22.: Unmodified TCP with DRR at Switches Goodput vs Number of Flows

In subsection 1, it was pointed out that lack of fairness at the very small timescales taken to transmit a single SRU over the link causes performance degradation. The problem in dealing with QCN rate control however, is the fact that the rates are not determined centrally by the CP, but by each RP individually, based on feedback from the CP. Even if all RPs started with the same rate, the packets originating from their respective flows would get sampled at different times, leading to different feedback values, and hence, different rates at different times.

One way of dealing with this problem is to make high feedback packets reach as many flows at once as possible. While it is not possible to send feedback for a packet to unrelated flows, if sampling is done very rapidly during times of heavy congestion, the delay between feedback calculation for different flows is minimized (and hence, flows get feedback packets that are closer in feedback values, as well as in time). Stepping up sampling during congestion could possibly help in increasing fairness. The sampling modifications applied to QCN are discussed later in this thesis.

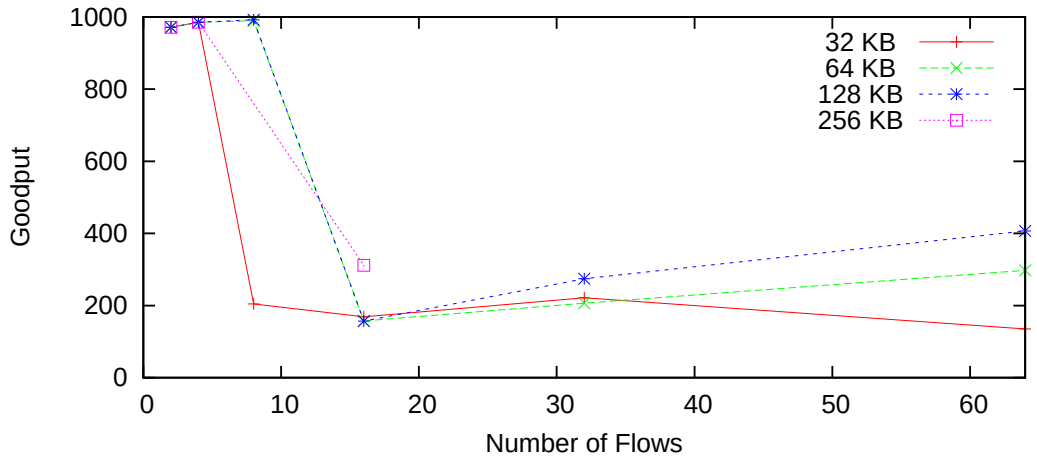



Fig. 23.: Unmodified TCP with SFQ with Hard Queue Partitioning at Switches Goodput vs Number of Flows

Another approach to the problem is to note that if flows are increasing their rates during AI (R_AI based increase) and decreasing the rates again after receiving negative feedback regularly, a very approximate estimation of the average range of flows would be the average time between negative feedback for a flow times R_AI/(AI timer period) (or R_HAI/(HAI timer period) for HAI). This estimate is not very accurate, but it gives an intuitive understanding of the significance of the R_AI value and timer period in the improvement of small time scale fairness among flows in the Incast setup. Both approaches were tried, but preliminary simulation results were good with R_AI reduction, while timer period increase caused reduction in goodput, and was not explored further. R_AI modifications are discussed later in this thesis. However, AI timer based modifications could also potentially be helpful for improving fairness among flows.

Further, in subsection 2, it was seen that TCP timeouts cause QCN to spuriously increase its flow rates. This can be dealt with in the two ways mentioned above as

well:

Sampling more during congestion would cause the RPs to cut down their rates more quickly during heavy congestion, when timeouts are most likely to occur. Once the rates are cut down sufficiently, performance stabilizes. Higher sampling can prevent timeouts even when a greater number of flows are present, thus delaying the onset of Incast.

Once timeouts do occur, it was seen that RP rates increase due to lack of negative feedback. With lower values of R_AI and R_HAI, these rates cannot be increased as quickly, and rates are lower at the next timeout, and so on, till rates converge around an optimal value. This approach too, does not completely mitigate TCP timeouts, it simply pushes them further away by allowing QCN to recover from timeouts occurring upto a certain number of flows.

QCN allocated flow rates are thus seen to increase significantly during TCP timeouts, and by the time the TCP flows begin transmitting again, due to the high rates allocated, they experience time-outs again. This leads to a behavior very similar to that of TCP Incast without QCN. Decreasing the rate increase during AI and HAI stage of QCN can push back the permanent onset of timeouts to a larger number of flows. It would be necessary to make QCN congestion aware though, so that rate increase is moderated only during periods of severe congestion, and not otherwise.

The proposed modifications to QCN, with the effect they have on QCN performance will be described in the coming chapter.

# CHAPTER IV

## EFFECTS OF MODIFICATIONS TO QCN TO MITIGATE INCAST COLLAPSE

In subsection 5 in the previous chapter, the reasons for the lack of performance of QCN in a TCP Incast setup were discussed. Further, several modifications to improve QCN performance were suggested. The modifications proposed to QCN, and the effect they have on QCN performance, are outlined below:

### A. QCN Timer Modifications

As discussed in the earlier chapter, rate increase of QCN flows during TCP timeouts can cause congestion again, and prevent the flow rates from converging properly. QCN performs self-increase by two mechanisms: i. Timer based increase and ii. Byte counter based increase. During a timeout, the flows are silent, and do not perform byte counter based rate increase. However, each time the RP timer period expires, self-increase is done, with the result that the flow rate reaches the original rate (at which negative feedback had been received), and begins to increase steadily beyond that, with an increment of R_AI each period. One of the modifications attempted was to increase the RP timer period itself.

The RP self-increase timer period was increased by a factor of 5 in preliminary simulations, and the overall link rates were seen to be much more uniform. However, with an increase in the number of flows, the performance degraded significantly: links were seen to have very low rates, and all of a sudden a link would have a sharp rate increase, which would then again get decreased to the average rate, while some other link would have an increase in link rate. Further timer based modifications have not been attempted so far.

## B. Sampling Modifications

In order to test the hypothesis of lack of sufficient feedback during severe congestion phases, QCN was initially modified to sample every packet. The results were seen to be much better, and collapse did not occur even for 32KB buffer size. Since every packet was sampled, the number of QCN feedback packets sent increased, but was just around 5 times as much as normal sampling. This can be explained by the fact that only negative feedback packets are sent, and not all sampled packets necessarily generate negative feedback (especially after flow rates stabilize to values that do not cause congestion).

However, as sampling every packet might not be necessary, especially during periods of less congestion. two different strategies which offer almost the same level of performance as sampling every packet, while sampling more packets only during congestion, are proposed. 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 less 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 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 no traffic 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. A congestion event is defined 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. The sampling frequency is scaled 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. An extra variable equivalent to 90% of the queue (QSC) also needs to be defined, 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 24. The performance is considerably improved compared to the QCN results shown earlier in Figure 10.

### 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 timeout before convergence, leading to spurious increases in QCN flow rates. However, with every packet being sampled, flow rates converge even

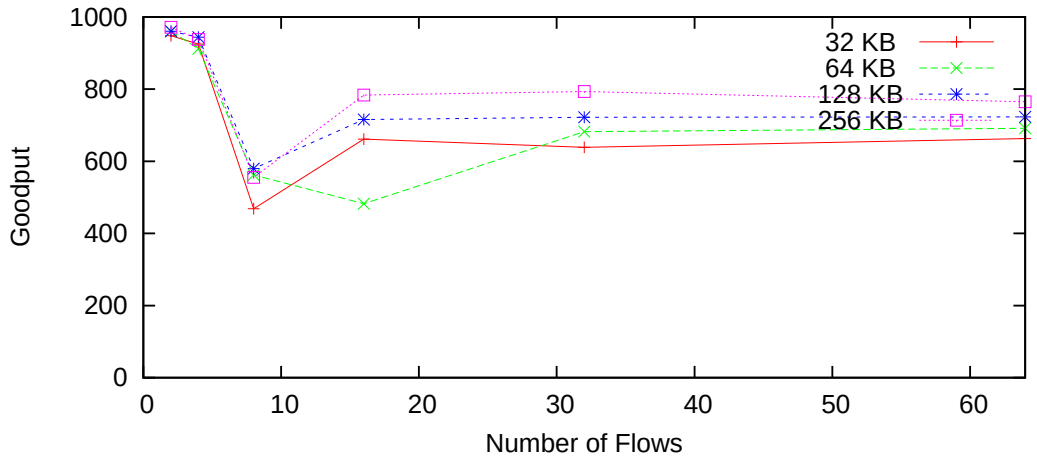



Fig. 24.: Congestion Memory Based Sampling in Incast Setup

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. The heaviest congestion level (7)) is expanded 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.

Table IV shows the sampling used in the unmodified QCN algorithm, while Table V shows how for the highest level of congestion (Feedback/8=7), 8 further sublevels have been created. The second columns in the tables show the average number of bytes the CP sends before sampling another packet, while the third columns show the number of packets sent assuming a packet size of 1500 Bytes. Note that though sampling frequency is modified, the feedback sent by the CP to the RP is unmodified, and is still 6 bits in length (0-63).

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.

Table IV.: Feedback Levels in Unmodified QCN

| **Feedback/8** | **Bytes to Send** | **Packets to Send** |
|---|---|---|
| **0** | 15000 | 100 |
| **1** | 75000 | 50 |
| **2** | 50000 | 33 |
| **3** | 37500 | 25 |
| **4** | 30000 | 20 |
| **5** | 25000 | 16 |
| **6** | 21500 | 14 |
| **7** | **18500** | **12** |

The performance of this strategy in the Incast setup is shown in Figure 25. The results are observed to be considerably better than the QCN results shown earlier in Figure 10.

## C. 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

Table V.: Extra Feedback Levels in Detailed Sampling (for Level 7 in Previous Table)

| **Feedback Level** | **Bytes to Send** | **Packets to Send** |
|---|---|---|
| **56** | 18500 | 12 |
| **57** | 15500 | 10 |
| **58** | 12500 | 8 |
| **59** | 9500 | 6 |
| **60** | 6500 | 4 |
| **61** | 4500 | 3 |
| **62** | 3000 | 2 |
| **63** | 1500 | 1 |

QCN creates enough variations that the variation in flow rates causes sub-optimal throughput for synchronized flows.

One way of dealing with this problem is to reduce the amount by which an RP increases its rate during congestion by making the self-increase rate (R_AI, R_HAI) congestion aware. The increase rate adjustment is done by dividing R_AI (and R_HAI) 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 and R_HAI.

The memory element is decremented by 1 each time the RP timer expires, leading to a slow increase in rate with recovery from congestion.

In terms of implementation cost, an extra counter is required in every Reaction

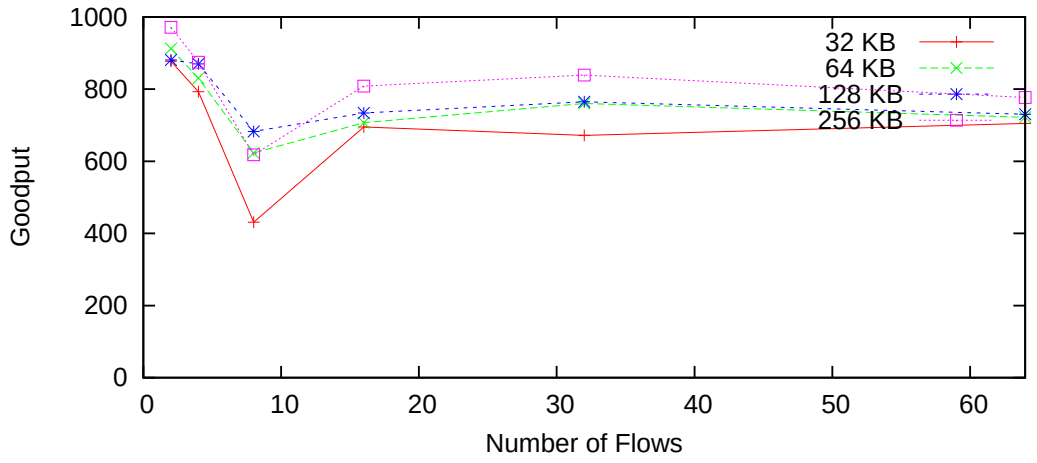



Fig. 25.: Detailed Sampling in Incast Setup

Point. Further, logic to divide target rate increase 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 and R_HAI: 50Mbps (or whatever values are 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.

To reduce false positives, the memory element is increased by the feedback amount only during the first cycle of fast recovery (the same strategy is employed for changing target rate in the QCN pseudocode).

The performance of this strategy in the Incast setup is shown in Figure 26. The results are observed to be considerably better than the QCN results shown earlier in Figure 10. Throughput still drops to below 300Mbps at 32 flows for 32KB buffer size and 64 flows for 64KB buffer size, but overall performance improves.

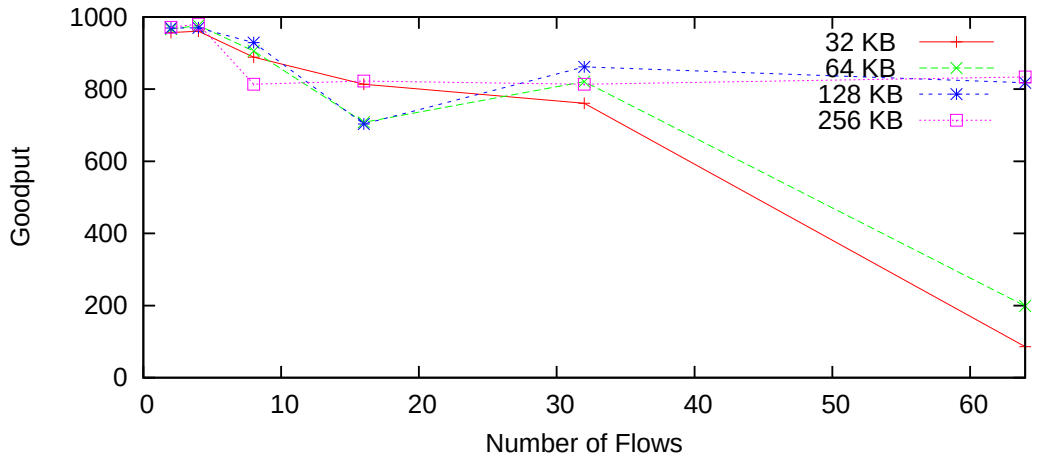



Fig. 26.: Adaptive R_AI in Incast Setup

## D. Combining CP and RP Modifications

Simulations were also run 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 27 and 28. 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.

The number of feedback messages sent in various systems with different modifi-

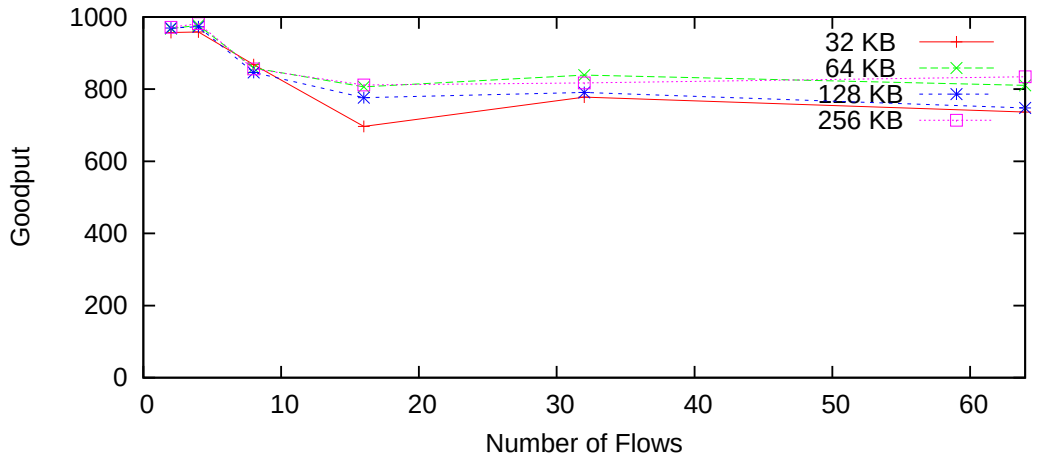



Fig. 27.: Adaptive R_AI with Congestion Memory Based Sampling in Incast Setup

cations to QCN are shown in Table VI. 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 a significantly smaller number of feedback messages than when sampling techniques are employed by themselves.

The average number of timeouts experienced by each flow in various experiments are shown in Table VII. 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. Note that the number of timeouts is much higher with Modified TCP with 10ms minrto after timeouts start occurring. However, as reducing the minimum RTO of TCP aims to reduce the penalty of TCP timeouts on flow rates than avoid timeouts, this still yields performance near 900 Mbps, even with 127.69 timeouts per flow at 64 flows.

Further, the total number of seconds spent without transmitting any data (over

Table VI.: QCN Feedback Packets Generated for: 128 KB Buffer Size

| **Flows** | **I** | **II** | **III** | **IV** | **V** | **VI** | **VII** |
|---|---|---|---|---|---|---|---|
| **2** | 4740 | 18969 | 7282 | 11716 | 3357 | 3337 | 8905 |
| **4** | 5535 | 10985 | 10514 | 10299 | 4454 | 5920 | 4853 |
| **8** | 6530 | 15996 | 10613 | 15386 | 7879 | 8762 | 8687 |
| **16** | 8844 | 21786 | 15494 | 19894 | 10840 | 12150 | 13444 |
| **32** | 14551 | 46670 | 30779 | 42621 | 21912 | 20146 | 28112 |
| **64** | 21846 | 155051 | 156161 | 121382 | 54405 | 36076 | 62553 |

I: Unmodified QCN
II: Sampling Every Packet
III: Congestion memory based sampling
IV: Detailed Sampling
V: Adaptive R_A
VI: II + IV
VII: III + IV

Table VII.: QCN Simulation Timeouts per Flow: 128 KB Buffer Size

| **Flows** | **I** | **II** | **III** | **IV** | **V** | **VI** | **VII** |
|---|---|---|---|---|---|---|---|
| **16** | 91.69 | 0.12 | 0 | 0 | 0.06 | 0 | 0 |
| **32** | 106.66 | 0.44 | 0 | 0 | 0.96 | 0 | 0 |
| **64** | 127.69 | 9.28 | 0.22 | 0.28 | 4.31 | 0.21 | 0.22 |

I: Modified TCP with 10ms minrto
II: Unmodified QCN with modified TCP
III: Congestion memory based sampling
IV: Detailed Sampling
V: Adaptive R_AI
VI: II + IV
VII: III + IV

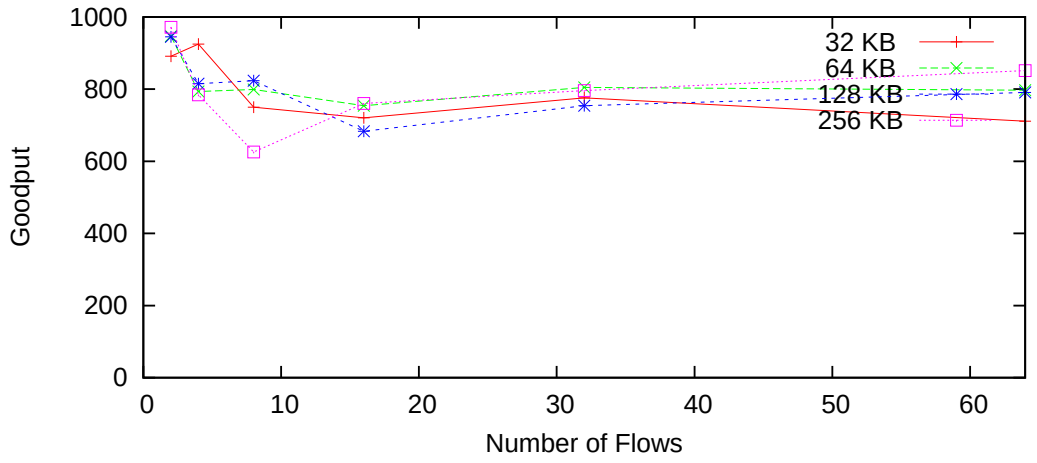



Fig. 28.: Adaptive R_AI and Detailed Sampling in Incast Setup

a total simulation time of 20 seconds) upto 64 flows was found to be 0 for both combinations of sampling and adaptive R_AI strategies, as well as for congestion memory based sampling. 5 seconds were spent without transmission for 32KB buffer size, 64 flows, for detailed sampling, while 9 seconds were spent without transmission for 32KB buffer size, 64 flows with only adaptive R_AI. Adaptive R_AI alone also had 1 second without transmission with 64KB buffer size and 64 flows. Overall, the performance of all modifications exceed those of unmodified QCN.

# CHAPTER V

## PERFORMANCE OF TCP AND QCN IN A MIXED PROTOCOL ENVIRONMENT

As seen in section D of the previous chapter, 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 27 and 28). However, as can be seen in Figure 9, 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 data center switches, and any performance problems due to the switches implementing QCN will lead to the performance issues seen in Chapter 3. 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.

The simulation setup was modified to add one high bandwidth UDP CBR flow (400Mbps) from an additional node to the client node, and the simulations were run both with and without QCN, and both with and without a modified (reduced minrto) TCP.

The performance of TCP (Figure 29) undergoes collapse even earlier than in the simple setup, with the number of servers supported per buffer size halved. Simulation results show that the setup with modified TCP with minrto 10ms only (no QCN)

yielded 400Mbps to the UDP flow (Figure 30), and fairly utilized the remaining bandwidth with 256KB buffer size. The performance was slightly worse with lower buffer sizes, but this performance would likely also nearly be upto 60% link utilization with smaller minrto values as proposed in [17].

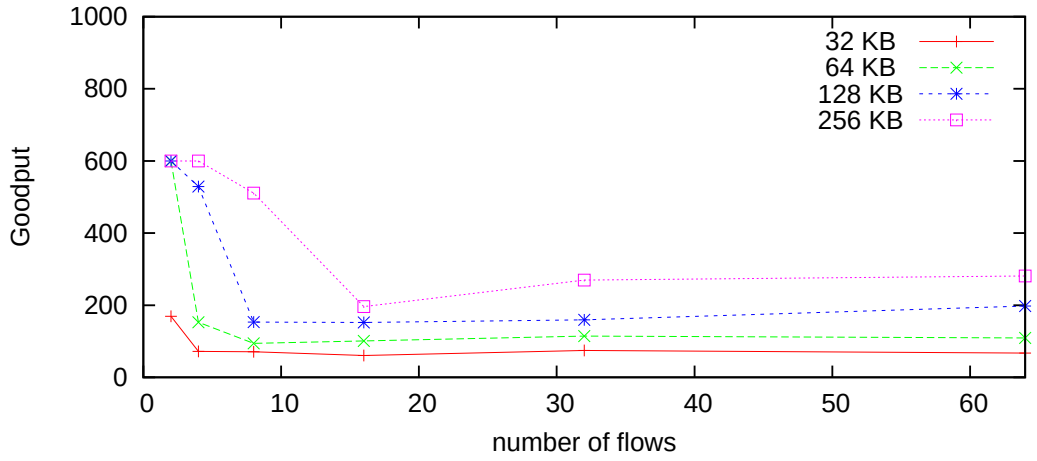



Fig. 29.: Performance of Unmodified TCP in Mixed Protocol Setup

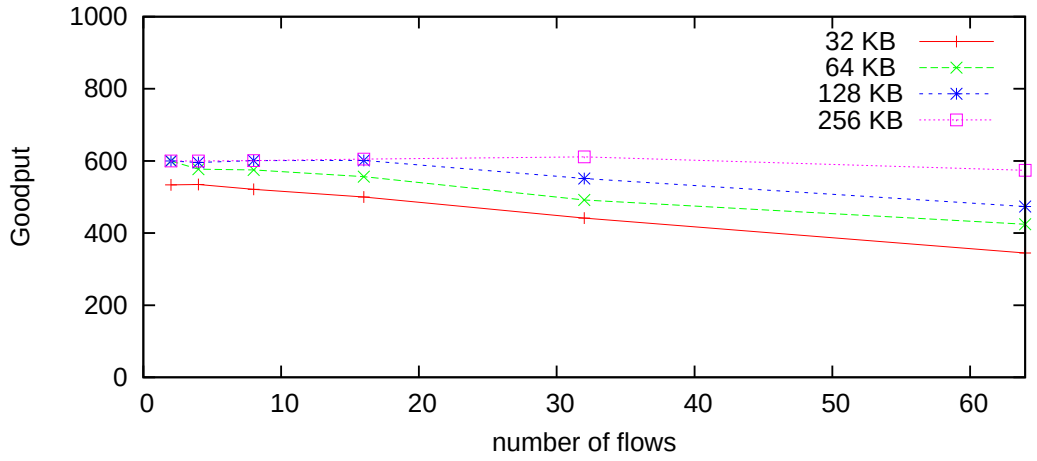



Fig. 30.: Performance of TCP with 10ms Minrto in Mixed Protocol Setup

The performance of unmodified QCN (Figure 31) 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 the goodput graph of unmodified TCP above QCN with the proposed modifications has only been shown. Adaptive R_AI + detailed sampling, however is seen to perform badly at 32 KB buffer size and 64 flows however (200Mbps throughput), though higher buffer sizes do well.

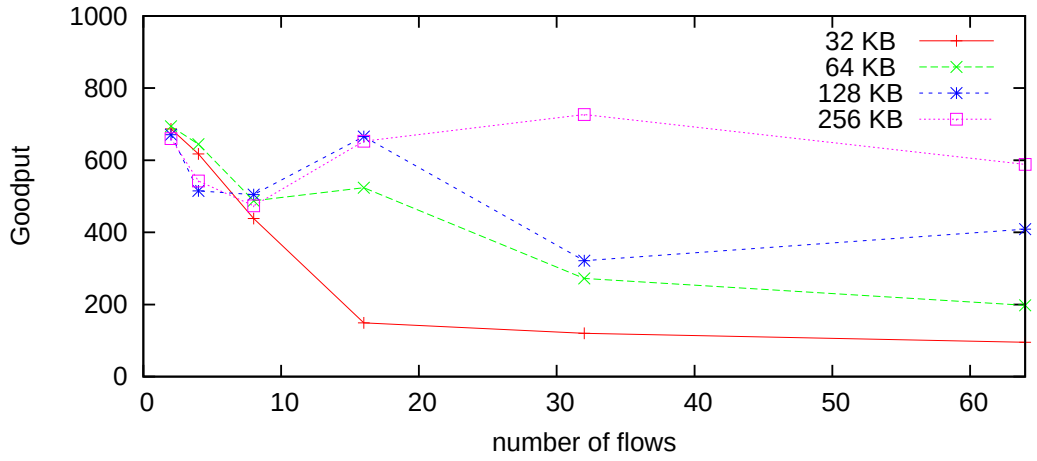



Fig. 31.: Performance of Unmodified QCN in Mixed Protocol Setup

With a modified QCN (Figures 32 and 33), 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. This is clearly an improvement over TCP with modified minrto, where fairness is improved among TCP flows, but the problem of competing with aggressive non-TCP flows still exists. One of QCN's main strengths is performing well in such environments, and while it does have an advantage over TCP in this regard, as seen from the performance of unmodified QCN above, modifying QCN helps greatly in a barrier synchronized flow setup.

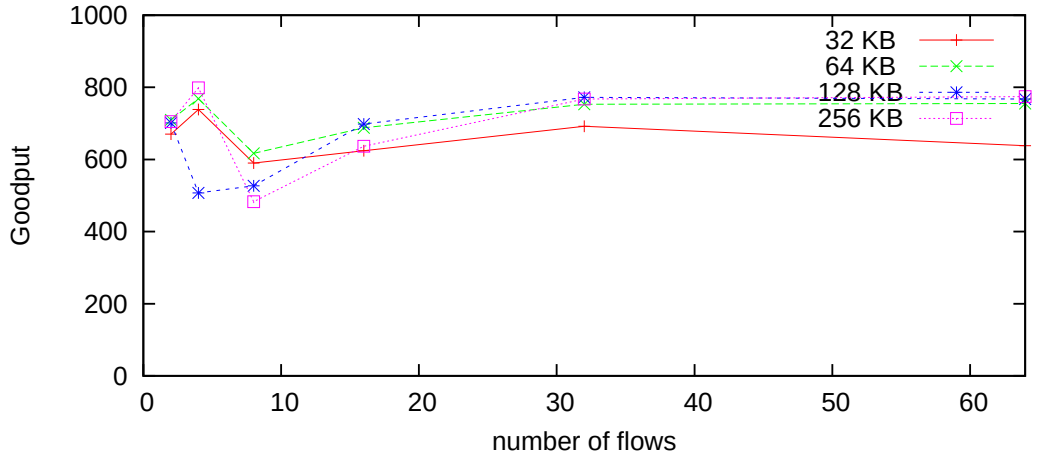



Fig. 32.: Performance of QCN with Adaptive R_AI and Congestion Memory Based Sampling in Mixed Protocol Setup

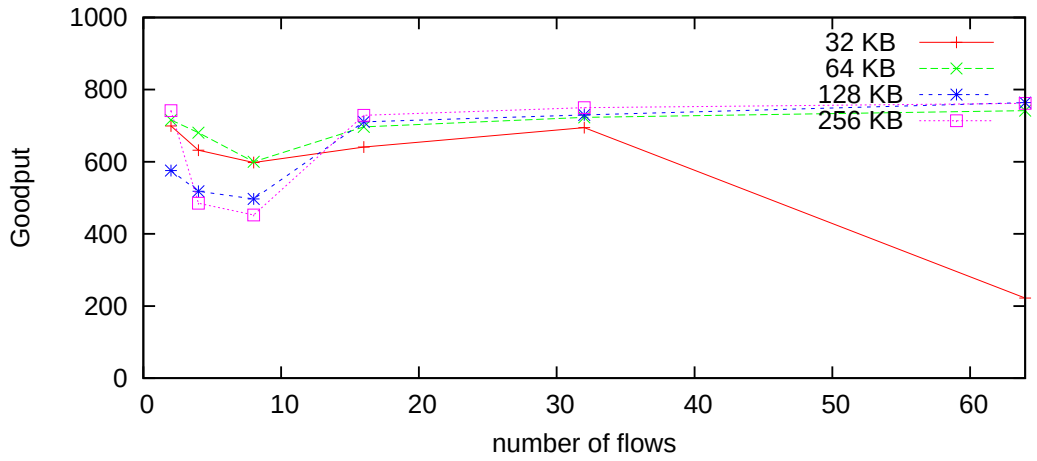



Fig. 33.: Performance of QCN with Adaptive R_AI and Detailed Sampling in Mixed Protocol Setup

# CHAPTER VI

## PERFORMANCE OF MODIFICATIONS IN QCN BASELINE SIMULATIONS

### A. Overview

Even though QCN with the proposed modifications performs much better than unmodified QCN in a TCP Incast setup, the effects this has on QCN performance in other workloads where QCN is intended to operate should be examined. The IEEE working group has a set of baseline simulations intended to be used for examining QCN performance in various setups. Four of the QCN baseline simulations have been examined as faithfully as possible using the ns-2 network simulator, with some minor differences in implementation. The performances of both QCN without modifications, as well as the modifications we have proposed have been examined, and the results will be discussed in this chapter. The factors that are especially important are responsiveness to congestion, queue stability, recovery from congestion, and fairness among flows.

When a congestion event occurs, the rate of any offending flows should quickly be adjusted in response to the congestion. Further, given that other conditions are similar among the flows, each should get a fair share of the bandwidth of the congested link. Any innocent flows (i.e. flows that do not contribute to the congestion occurrence) should not be penalized because of the congestion event. When the congestion event disappears, the rates of the concerned flows should be accordingly adjusted to achieve maximum link utilization in the uncongested link. The performance of QCN with modifications should be better than, or at least as good as, that of unmodified QCN in these aspects.

The four baseline simulations outlined in [22], [23], and [24] were implemented

in order to evaluate the performance of the proposed modifications against that of unmodified QCN. However, due to ns-2 implementation difficulties, virtual output queuing (VOQ) could not used in the switches in the baseline simulations. The default output queuing mechanism of ns-2 was used instead. Node traffic was generated using Constant Bit Rate (CBR) sources, as initial simulations showed CBR and poisson (exponential traffic) sources performing similarly.

The difference in the queuing model however,should not make a major difference in the in the flow traffic distribution. The other simulation parameters are as described in the original simulations.

The results of the simulations will be summarized in this section. The details of each baseline simulation, and relevant results for the simulations will be presented in the following sections.

The queue behavior of the simulations were somewhat different from baseline simulations. The queue lengths fluctuated between 0 and a peak value (depending on each simulation), while the average value was close to the equilibrium point of QCN. This is possibly because of the queuing strategy employed (our simulations use simple Droptail Queuing, while the cited simulations use Virtual Output Queuing), or other simulation parameter differences (e.g. the latency of the simulations was not clear, and 1 microsecond link-link latency was used). However, the traffic behavior was very similar to that shown in the slides. For this reason, comparison is done with results obtained in our simulations instead of with the results presented in the slides, while obtained results are compared with that in the slides as a general sanity check (traffic behavior should be similar, and queue length should be similar). Sampling strategy was observed to be very important in queue behavior, traffic, and number of drops. On the other hand, adaptively changing R_AI and R_HAI was seen to introduce a minimal amount of latency into recovery from congestion (will be examined in more

detail in relevant sections).

A few general observations based on the simulations are listed below, though only results for unmodified QCN, QCN with adaptive R_AI + congestion based memory sampling and QCN with R_AI and detailed sampling are presented for brevity. It was seen that:

1. The performance of the QCN based on the older pseudocode is slightly better than the implementation based on pseudocode version 2.2 (which we shall continue referring to as unmodified QCN) in terms of queue drops and queue lengths.

2. Congestion memory based sampling does not have much of an effect on QCN performance in the baseline simulations as compared to unmodified QCN performance.

3. R_AI modifications have only a slight effect on QCN performance: improving it somewhat, but not much. The time to recover from congestion is slightly longer.

4. Detailed sampling results are very similar to the results obtained from QCN based on [19].

5. 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).

The details of each simulation will be presented in the following sections. The QCN parameters used in the simulations are listed in Appendix A.

## B. Baseline 1

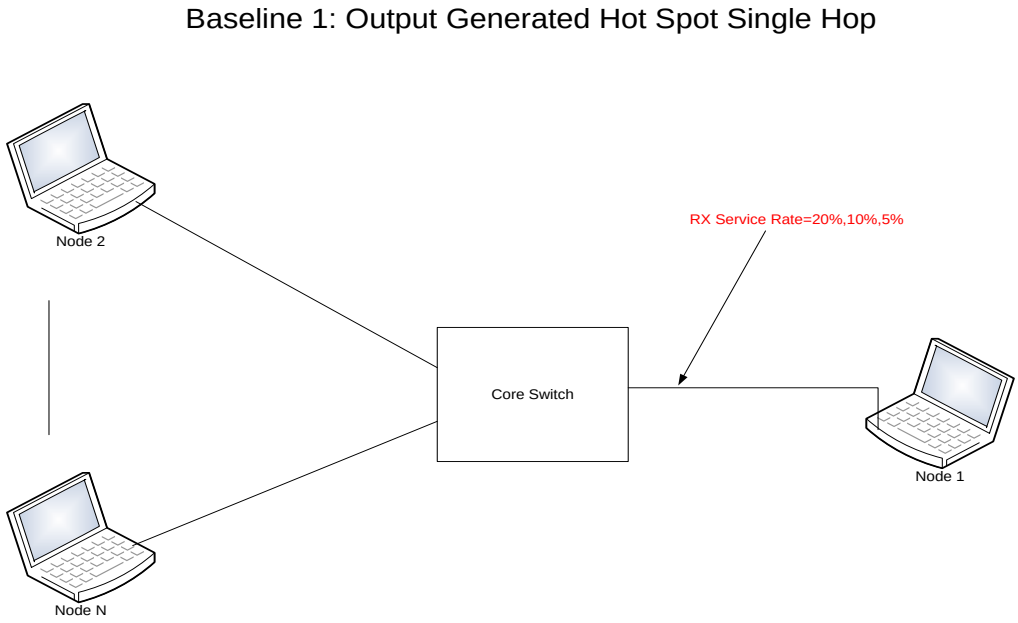



Fig. 34.: QCN Baseline Simulation 1 (Based on [23])

### 1. Workload

All Nodes (10): Uniform Distribution, load = 8.5 Gbps

Node 1 Service Rate = 1 Gbps

One Congestion Point

Hotspot (2 Gbps, 1 Gbps, and 0.5 Gbps output generated service rate scenarios):

Degree: 9, Severity: 8.5:

Duration: 80ms from time=10 to 90ms

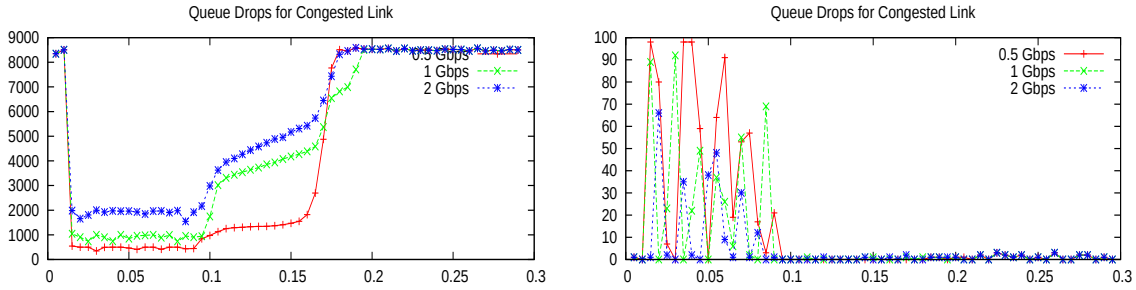
The setup for Baseline 1 is shown in Figure 34 (based on [23]). In the first baseline simulation, a congestion event is created starting at 10ms and ending at 90 ms. During this time, the service rate of the bottleneck link (Core Switch-Node 1) goes down from 10Gbps to 2, 1, and 0.5 Gbps, depending on the simulation. For each simulation, the throughput of the bottleneck link, the queue length of the link, and the total number of drops are observed. A steady queue length during congestion implies stability during congestion, while oscillations could lead to overutilization/underutilization of the link in severe cases.

The number of drops and time for the queue length to stabilize indicate the response to congestion events, while the rise of bandwidth with time after congestion is over indicates the recovery of QCN from congestion.

Behavior of unmodified QCN and QCN using adaptive R_AI+congestion based memory sampling are very similar: the queue length reaches full capacity several times, though the average length is quite stable. There are drops when the bottleneck occurs, and the number of drops is constant afterwards (graph not shown for brevity). For adaptive R_AI + detailed sampling, the number of drops is far less, and the queue occasionally spikes up, but the overall queue length is much less than that with only QCN or QCN + adaptive R_AI + congestion based memory sampling. This behavior is very similar to the results observed if sampling outlined in [19] is used. The actual number of drops are shown below with the figure captions.

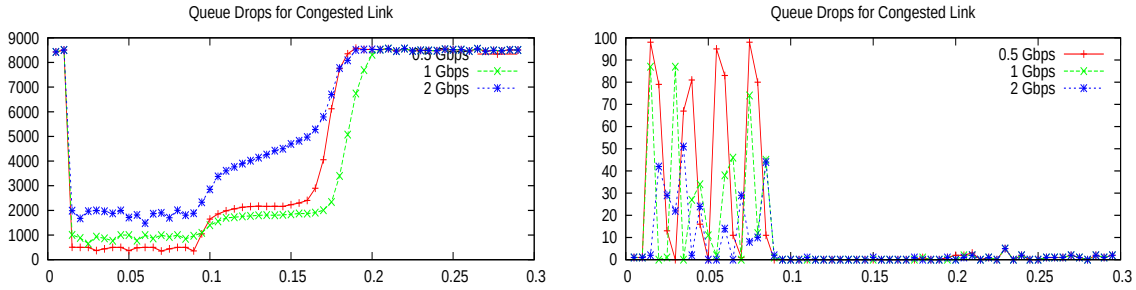
The link utilization of the congested link is not much for the first 25 ms, after which the congested link is utilized in adaptive R_AI + detailed sampling. After the congestion event goes away, the link rate picks up more slowly as compared to unmodified QCN for 1 Gbps for both modifications and 2 Gbps as well for adaptive R_AI + detailed sampling, but link rates rise steeply during the HAI stage, leading to full link utilization within comparable times of that of unmodified QCN.

As seen in Figures 35, 36, and 37, the fastest recovery from congestion occurs when the bottleneck is limited to 2Gbps, and QCN with adaptive R_AI+Detailed sampling recovers the fastest, while unmodified QCN and QCN with adaptive R_AI+congestion memory based sampling perform very similarly. The queue lengths are also similar for the latter two, while the former has smaller queue lengths. Number of drops are similar for the latter two, while for the former, they are very low. Overall, results indicate that the proposed modifications do not have a negative effect on QCN behavior.



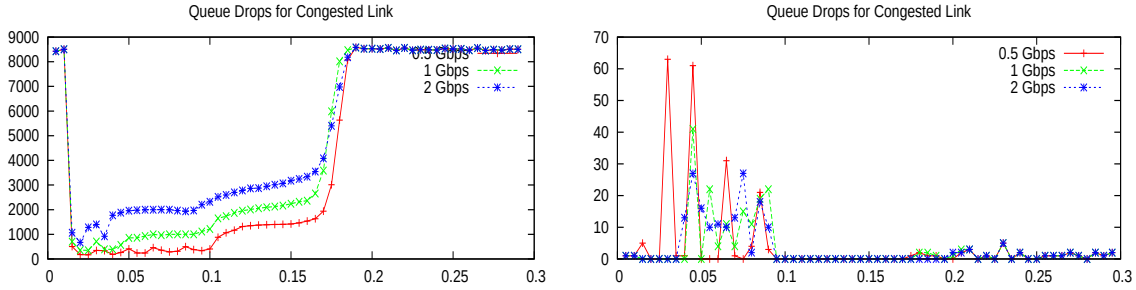

(a) Throughput (b) Queue Length

Fig. 35.: Normal QCN (Drops: 0.5GB: 1207 1GB: 884 2GB: 618)




(a) Throughput (b) Queue Length

Fig. 36.: QCN + Adaptive R_AI + Congestion Memory Based Sampling (Drops: 0.5GB: 1157 1GB: 893 2GB: 632)




(a) Throughput (b) Queue Length

Fig. 37.: QCN + adaptive R_AI + Detailed Sampling (Drops: 0.5GB: 99 1GB: 71 2GB: 45)

## C. Baseline 2

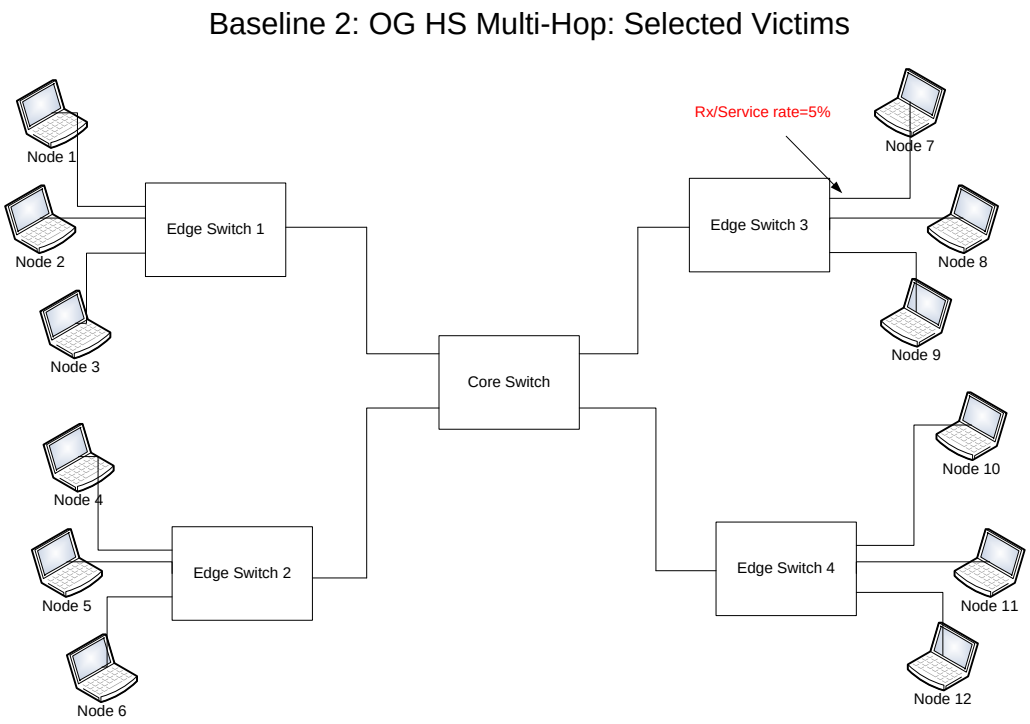



Fig. 38.: QCN Baseline Simulation 2 (Based on [22])

### 1. Workload

All: Uniform Distribution traffic (background traffic)

Nodes 1-6: 25% (2.5Gbps) Nodes 7-10: 40% (4Gbps)

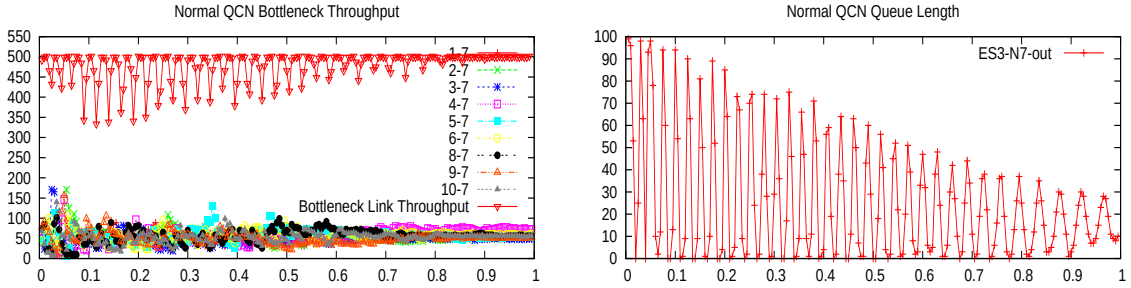
Primary Hotspot:

Node 7 service rate = 5% (Rx only)

If saturation tree spreads = 5 congestion points in total

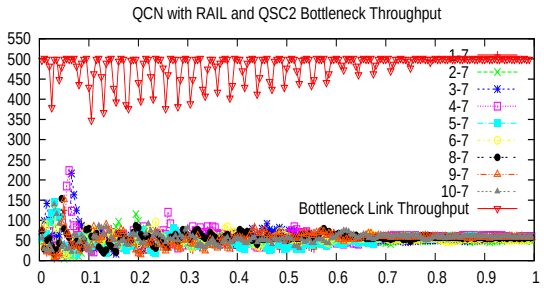
The setup for Baseline 2 is shown in Figure 38 (based on [22]). In Baseline 2, the bottleneck link receives uniform traffic from other nodes, but its service rate is only 5% of the actual link capacity, causing a bottleneck to be formed. As seen in

Figures 39, 40, and 41, queue length gradually settles down for unmodified QCN and QCN + adaptive R_AI + congestion based memory sampling, while it varies between 0 and a peak of 45 packets for QCN + adaptive R_AI + detailed sampling. For the first two, rates gradually converge to rates that are very close to each other, and full link capacity is utilized, (the label -7 in the graphs, signifying traffic from all sources to node 7), while for the latter strategy, rates converge together more quickly, but constantly go up and down a small range. The link utilization varies between 70% to full utilization, giving an average utilization of around 85% link utilization. Less variation is anticipated with the use of Virtual Output Queuing, as outlined in the simulations outlined in [22], [23], and [24]. Overall, traffic behavior and queue behavior is not negatively affected by proposed modifications.

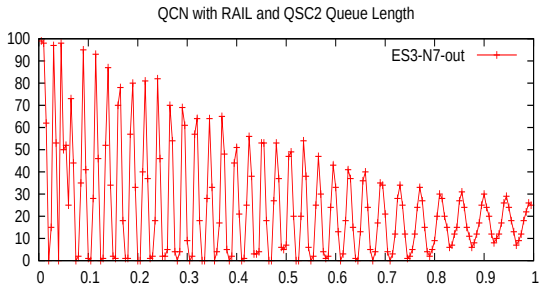



(a) Throughput (b) Queue Length

Fig. 39.: Normal QCN (Drops: 1196)

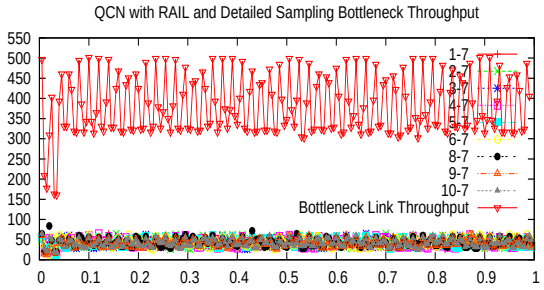



(a) Throughput

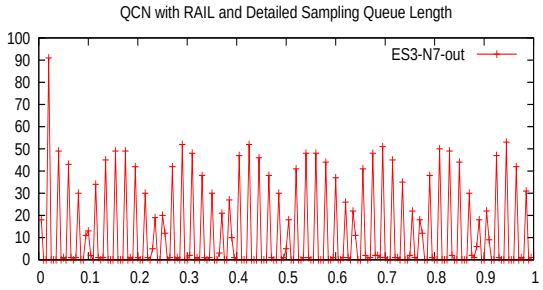



(b) Queue Length

Fig. 40.: QCN + Adaptive R_AI + Congestion Memory Based Sampling (Drops: 1118)




(a) Throughput




(b) Queue Length

Fig. 41.: QCN + Adaptive R_AI + Detailed Sampling (Drops: 94)

## D. Baseline 3

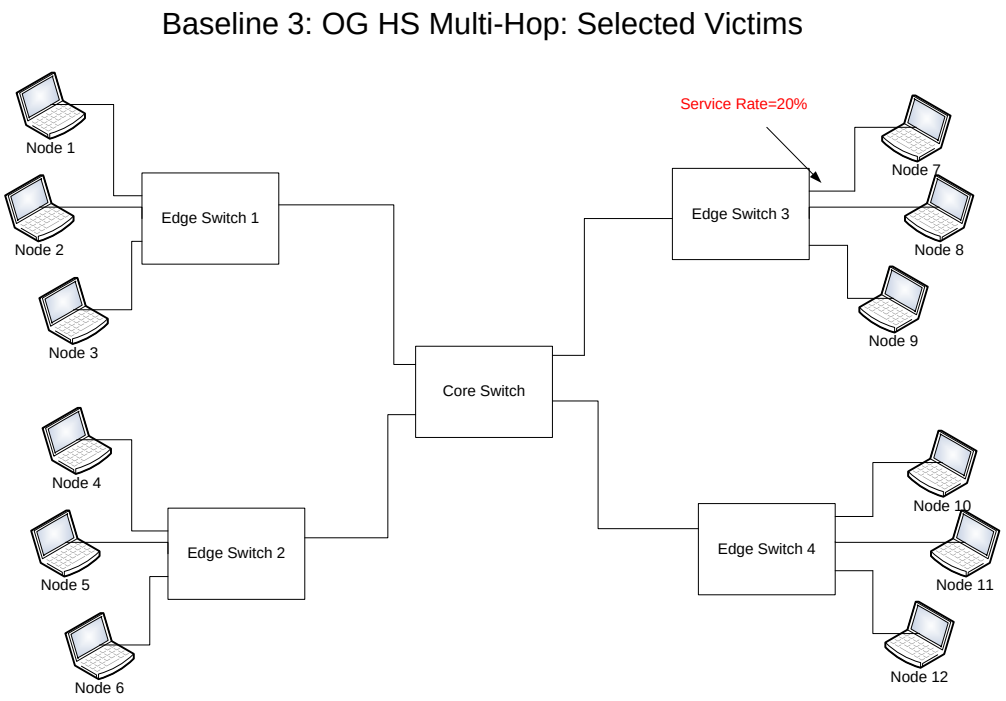



Fig. 42.: QCN Baseline Simulation 3 (Based on [22])

### 1. Workload

Four culprit flows of 2 Gb/s each from nodes 1, 4, 8, 9 to node 7 (hotspot)

Three victim flows of 7 Gb/s each: node 2 to 9, node 5 to 3, node 10 to 6

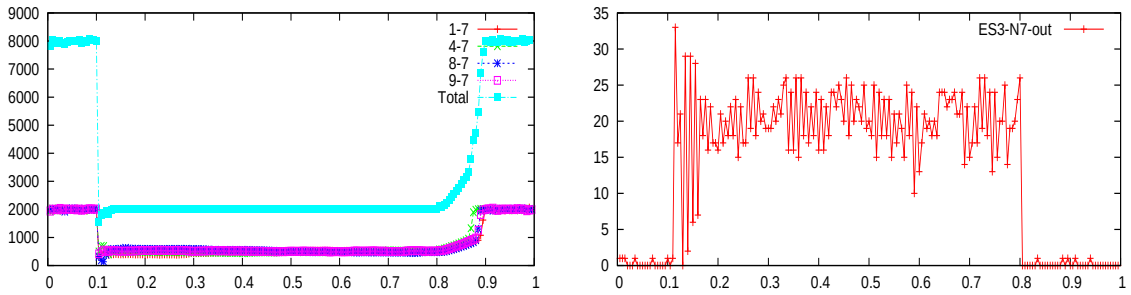
Node 7 service rate = 20%

Five congestion points, all switches and flows affected

Fair allocation provides 0.5 Gbps to all culprits and 7 Gbps to all victims

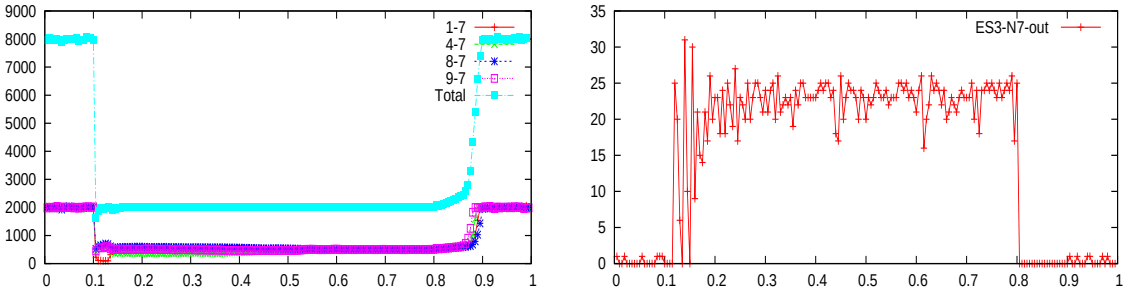
The setup for Baseline 3 is shown in Figure 42 (based on [22]). In the third baseline simulation, the four culprit flows mentioned above can cause problems for the three victim flows if PAUSE is issued (which we are not simulating). However, the

allocation between the culprit flows should also be fair, and each should get 0.5Gbps rate allocation. As seen in Figures 43, 44, and 45, queue behavior is very similar for all unmodified QCN and the proposed modifications. The number of drops in unmodified QCN and QCN + adaptive R_AI + congestion memory based sampling are very similar, while there are no observed drops for QCN + adaptive R_AI + detailed sampling. All three simulations are seen to allow full link utilization of the bottleneck during congestion (0-1-0.9s), and to recover from congestion at similar intervals. Fairness among flows is observed, with slightly more variation in QCN + adaptive R_AI + detailed sampling, in the simulations carried out, but the amount of variation is not seen to be very significant. Overall, both proposed modifications perform well, and do not seem to affect QCN performance much.



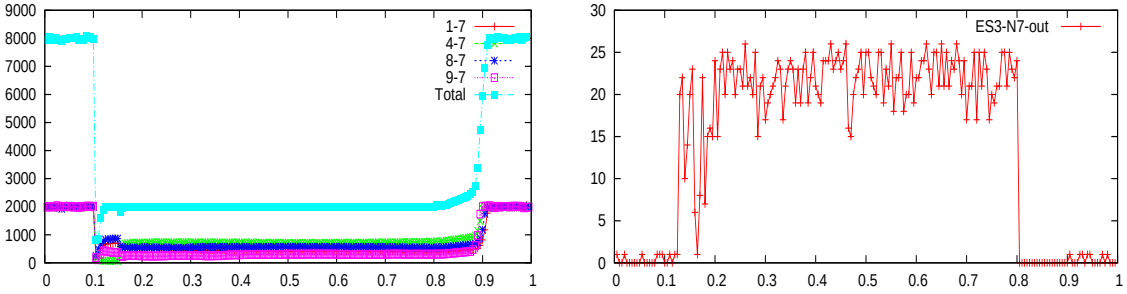

(a) Throughput (b) Queue Length

Fig. 43.: Normal QCN (Drops: 189)




(a) Throughput (b) Queue Length

Fig. 44.: QCN + Adaptive R_AI + Congestion Memory Based Sampling (Drops: 184)




(a) Throughput (b) Queue Length

Fig. 45.: QCN + Adaptive R_AI + Detailed Sampling (Drops: 0)

## E. Baseline 4

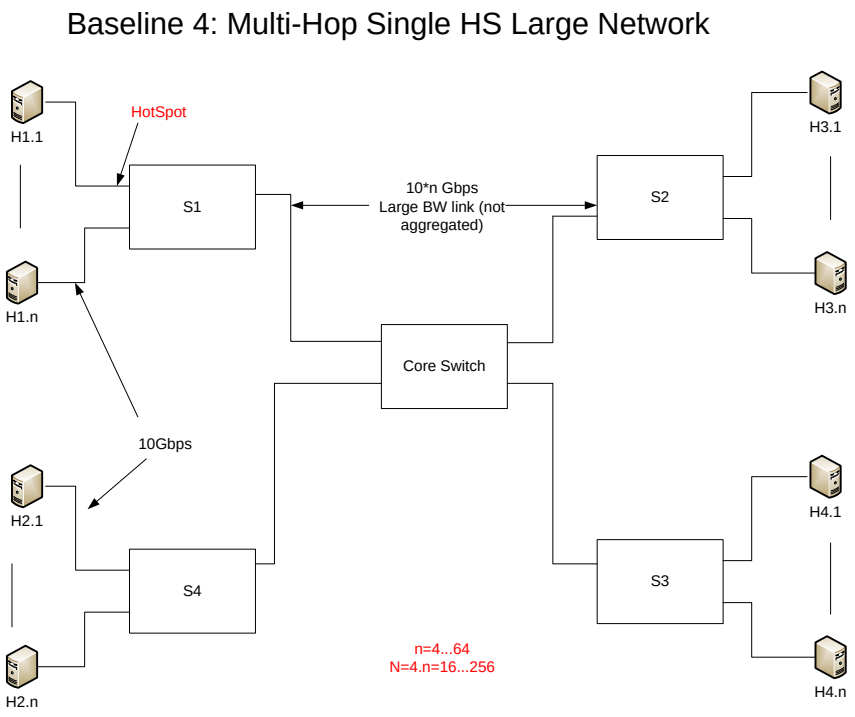



Fig. 46.: QCN Baseline Simulation 4 (Based on [24])

### 1. Workload

Load: H1.1 – H4.n $\lambda$ = 85%, Skewed Uniform

H1.1 is targeted with 2 $\lambda$

All other nodes with $\lambda$ (N - 2)/(N - 1)

Congestion Point:

Node H1.1

HS degree = N

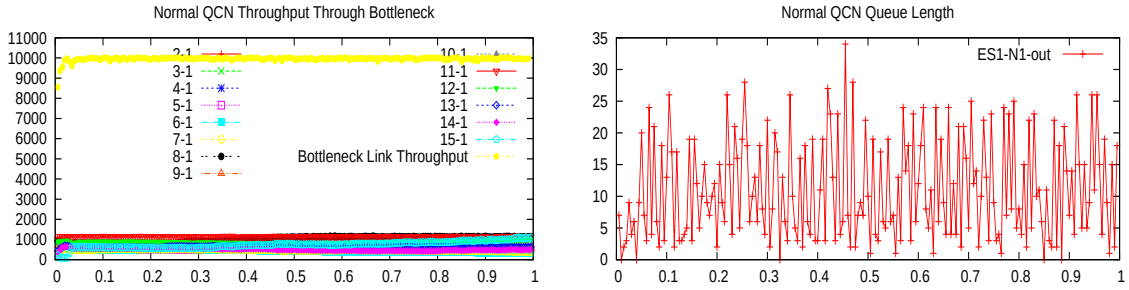
HS severity = 1.7:1

**Note:** For our purposes, only n=4 (i.e. N=16) has been simulated.

The setup for Baseline 4 is shown in Figure 46(based on [24]). In the fourth baseline simulation, there is no drop in the link rate, and instead, the bottleneck link is targeted with excess traffic. The stability of the queue is examined, as well as the link utilization of the bottleneck and the fairness among flows. A small variation is observed among flow rate allocations, but the fairness is much better than that with just UDP and no flow control. As seen in Figures 47, 48, and 49, queue length behavior is quite similar in all three simulations. However, it is seen in the figures that there is some variation among flows, with some flows transmitting at full capacity, and others transmitting at as low as 300Mbps for unmodified QCN, 400Mbps for QCN + adaptive R_AI + congestion memory based sampling and QCN + adaptive R_AI + detailed sampling. The results presented in [24] show variations in flow rates as well, with low rates being around 500Mbps, and high rates being around 1000Mbps, with not much variation. However, in our simulations, it was observed that some flows would constantly transmit at around 1100Mbps, while the QCN rates allocated to them were much higher (e.g. 3Gbps). This implied that even when QCN rates were cut down, these flows would transmit at much less than the total rate allocated, so would continue to enjoy full rate transmission.

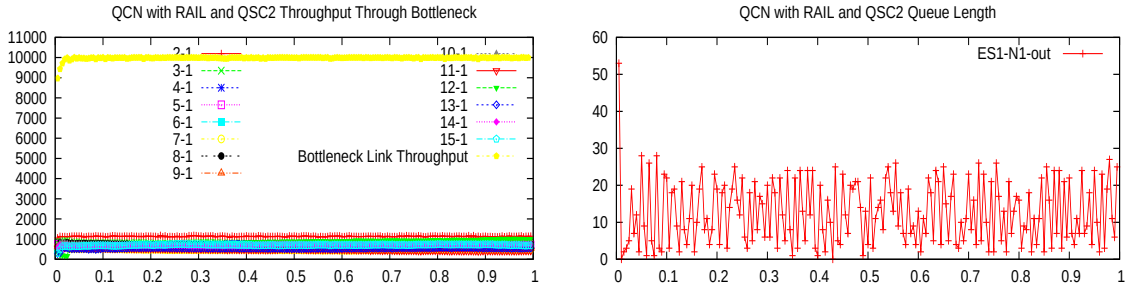
Due to rates being cut down, queue lengths are not very high, and high feedback is not received. This results in flows with lower allocated rates to be unable to increase rates by much, while flows with high allocations continue transmitting at high rates (and their higher rate enables more frequent byte based rate increase). This problem is one of fairness, and is seen to be present in [24], though to a lesser extent, possibly because of Virtual Output Queuing. However, as the range of flow variations is seen in all three simulations while the queue length fluctuates around the equilibrium point, the proposed modifications are seen to perform similarly to unmodified QCN. Once again, drops for QCN + adaptive R_AI + detailed sampling are seen to be much less

than those for unmodified QCN and QCN + adaptive R_AI + congestion memory based sampling.



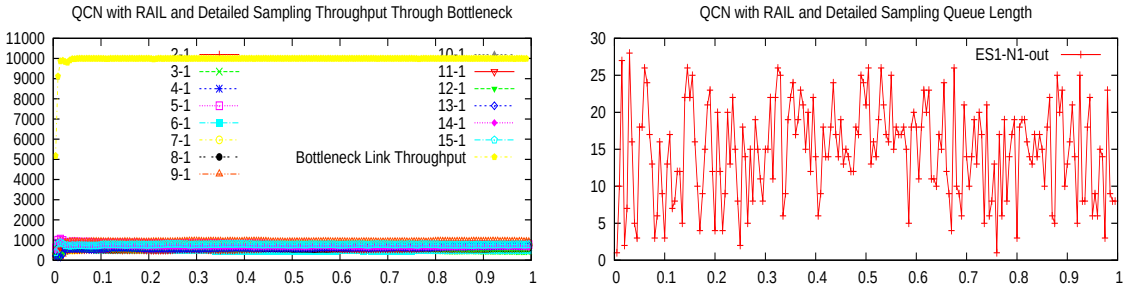

(a) Throughput (b) Queue Length

Fig. 47.: Normal QCN (Drops: 382)




(a) Throughput (b) Queue Length

Fig. 48.: QCN + Adaptive R_AI + Congestion Memory Based Sampling (Drops: 389)




(a) Throughput (b) Queue Length

Fig. 49.: QCN + Adaptive R_AI + Detailed Sampling (Drops: 17)

# CHAPTER VII

## CONCLUSIONS AND FUTURE WORK

QCN is being investigated as a congestion control mechanism to be implemented in data centers, where clustered storage and mixed protocol environments are both likely to be found. It is desirable to use switches with QCN to support various data center 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 data center 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.

The causes of QCN sub-optimal performance in a basic TCP Incast setup have been examined, and several modifications that do not affect QCN behavior in normal setups, and are not very difficult/expensive to incorporate have been suggested. Though these modifications do offer good link utilization, further work on improving QCN performance can be done. Possibility of timer based modifications and other mechanisms for improving QCN fairness in the TCP Incast setup, where not only aggregate fairness over time is important, but fairness over very short time intervals is also important, is one such possible area of study.

# 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, Last Accessed: March 16, 2010.

[3] "Lustre File System," http://wiki.lustre.org/index.php/Main_Page, 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, 2009, ACM, 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, 2007, pp. 361–372.

[8] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on router buffer sizing: recent results and open problems," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 2, pp. 34–39, 2009.

[9] 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, 2008, pp. 1–14, http://portal.acm.org/citation.cfm?id=1364813.1364825.

[10] 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.

[11] "Ethernet flow control," http://en.wikipedia.org/wiki/Ethernet_flow_control, Last Accessed: March 16, 2010.

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

[13] D. Bergamasco, "Data center ethernet congestion management: backward congestion notification," in *IEEE 802.1 Meeting*, 2005, http://www.ieee802.org/1/files/public/docs2005/new-bergamasco-backward-congestion-notification-0505.pdf.

[14] J. Jiang, R. Jain, and C. So-In, "Forward explicit congestion notification (FECN) for datacenter ethernet networks," in *Proc. SPECTS 2008: Performance evaluation of computer and telecommunications systems*, Edinburgh, UK, 2008, pp. 542 – 546.

[15] R. Pan, B. Prabhakar, and A. Laxmikantha, "QCN: Quantized congestion notification," *Online*, 2007, http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-description.pdf.

[16] A. Kabbani, R. Pan, B. Prabhakar, and M. Seaman, "QCN: algorithm for p-code," *Online*, 2007, http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-with-timer-0711.pdf.

[17] 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.

[18] 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.

[19] R. Pan, "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.

[20] J. M. McKenney, "Stochastic fairness queuing," in *IEEE INFOCOM*, San Francisco, CA, USA, 1990, pp. 733 – 740.

[21] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 4, pp. 231–242, 1995.

[22] 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.

[23] B. Atikoglu, A. Kabbani, R. Pan, and B. Prabhakar, "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.

[24] B. Atikoglu, A. Kabbani, R. Pan, and B. Prabhakar, "QCN: Benchmark simulations - scenario 4," IEEE, Tech. Rep., 2008, http://www.ieee802.org/1/files/public/docs2008/au-sim-rong-qcn-hai-0208.pdf.

# APPENDIX A

## QCN BASELINE SIMULATION PARAMETERS

The parameters used are listed below:

**Traffic**

Constant Bit Rate Source

Uniform destination distribution (to all nodes except self)

Fixed packet size = 1500 B

**Switch**

Output Queue with 100 packets queue size per output

**Adapter**

Token Bucket based rate limiting

One rate limiter per destination

Egress Buffer Size = 1500 KB

Ingress Buffer Size = Unlimited

The QCN parameters used in the baseline simulations are listed below:

**QCN**

W = 2.0

Q_EQ = 33 KB

GD = 0.0078125

Base marking: once every 150 KB

Margin of randomness: 30%

Runit = 1 Mb/s

MIN_RATE = 10 Mb/s

BC_LIMIT = 150 KB

TIMER_PERIOD = 15 ms

R_AI = 5 Mbps

R_HAI = 50 Mbps

FAST_RECOVERY_TH = 5

Quantized Fb: 6 bits

# VITA

Prajjwal Prasad Devkota received his Bachelor of Engineering degree in electronics and communications engineering from Birla Institute of Technology, Mesra, Ranchi, India, in 2002. He joined the Computer Engineering program under the Electrical Engineering Department of Texas A&M University in August 2007, and received his Master of Science degree in May 2010. His research interests are in computer networks and security.

Mr. Devkota can be reached at the Department of Electrical and Computer Engineering, Texas A&M University, 331E Wisenbaker, College Station, TX 77843-3259. His email is prajjwal@tamu.edu.