Improving Flow Completion Times in Data Center Networks

By
Ali Munir
2009-NUST-MS-EE-29

Supervisor
Dr. Saad Qaisar
NUST-SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Electrical Engineering (MS EE)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(March 2012)

Approval

It is certified that the contents and form of thesis entitled "**Improving Flow Completion Times in Data Center Networks**" submitted by **Mr. Ali Munir** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Saad Qaisar**

Signature: _____
Date: _____

Committee Member: **Dr. Ihsan Ayyub Qazi**

Signature: _____
Date: _____

Committee Member: **Dr. Syed Ali Khayyam**

Signature: _____
Date: _____

Committee Member: **Dr. Junaid Qadir**

Signature: _____
Date: _____

To my parents.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Ali Munir**

Signature: _____

Acknowledgments

First and foremost, I am immensely thankful to Almighty Allah for letting me pursue and fulfill my dreams. Nothing could have been possible without His blessings.

I feel proud the way my parents and younger siblings have shown patience and confidence in my abilities. They were always there to provide me with their full support in the difficult of times. This thesis may not have been possible without the sacrifices made by them.

My heartfelt thanks to my advisor Dr. Saad Qaisar for his kind support and guidance. I am also grateful to Dr. Andrew Lachlan for his useful feedback to improve quality of this work. I would also like to thank everyone at CoNNekt Lab, especially Haroon, Jaweria and Waqar for providing me a window outside of work and making my time at SEECS pleasant.

Lastly, this thesis would not have been possible without the expert guidance of Dr. Ihsan Ayyub Qazi, who has been a great source of inspiration for me during these years of research. I feel thoroughly indebted to him, his research vision has helped all along this strenuous path, right from the start to the completion of MS thesis.

Ali Munir

Table of Contents

List of Figures

List of Tables

Abstract

The popularity of online services such as web search, advertisement, social networking, and recommendation systems has lead to a proliferation of data center networks in the last few years. These services share a few common themes. First, they are soft real time nature. Consequently, the response time of tasks determines the quality of results, which in turn impacts operators revenue. Second, data centers typically employ a *Partition/Aggregate* workflow pattern, in which user requests are partitioned amongst layers of workers whose results are then aggregated. The returned responses from workers are often synchronized, leading to the *Incast* problem. In addition, data center applications generate a mix of traffic workloads. While most flows are short (<100 KB), a few long flows can saturate the path. Consequently, such flows can lead to large queueing delays and thus affect delay-sensitive traffic.

In this work, we leverage the above characteristics to design a protocol that seeks to reduce average flow completion time (AFCT) within data centers. To this end, first, we explore the efficacy of flow size awareness for reducing flow completion times in data centers. Second, we present the design, analysis and evaluation of a size aware delivery control protocol (SDC) which incorporates these features to to achieve small flow completion times while maintaining high network throughput and small queues. Third, we present evaluation of a router assisted capacity sharing (RACS) framework. Detailed performance analysis shows that RACS can improve AFCT of applications significantly, hence improve application throughput. Performance comparison with existing schemes like rate control protocol (RCP) and TCP-SACK shows that DC network performance can be significantly improved by leveraging flow size information at transport layer.

Chapter 1

Introduction

1.1 Motivation

The popularity of online services such as web search, advertisement, social networking, and recommendation systems has increased immensely in the last few years. Many of these services are hosted on data centers. Thus, besides Internet, data center infrastructure plays a major role in the quality of these services. Such services, constitute significant portion of traffic and is expected to increase in future [2, 3]. One of the key performance metric of such systems is average flow completion time (AFCT). Users care less about system efficiency and throughput or the latency of individual packets; they only care about their own flows.

Short FCTs also reduce the control loop delay of distributed applications interacting over the network. So it is perhaps surprising that almost all work on congestion control focuses on metrics such as throughput, bottleneck utilization and fairness. While these metrics are interesting - particularly for the network operator - they are not very interesting to the user. One of the most important building block of providing these services is the resource sharing and congestion control mechanisms employed within and outside data center networks. Currently, TCP is the most widely used congestion control protocol within and outside data centers [1, 4, 5].

Since its inception, congestion control over Internet has gained significant attention and as a result many new protocols have been proposed [5, 6, 7, 8]. However, when these protocols are applied within a data center network, a significant performance limitation is observed [9, 10]. This incapability has arisen due to diverse applications found inside a data center, high link

capacity, low latencies and data center architecture. Below we discuss characteristics of today's data centers, with a focus on aspects that influence the design of transport protocol.

Partition/Aggregate

Large scale web applications such as search and social networks achieve horizontal scalability by partitioning the task of responding to users amongst several worker machines [4]. In addition, services like MapReduce [11], Dryad[12], and Memcache [13] also follow this structure.

A consequence of the partition/aggregate workflow pattern is that responses from several servers frequently get synchronized at a router. This can cause large queue build and burst losses at a router that can lead to throughput collapse. This is known as the Incast problem [9]. Incast occurs frequently and has been observed in several data centers.

Mix of flow sizes

In addition, data center applications generate a mix of traffic workloads. While most flows are short (<100 KB), a few long flows can saturate the path, which suggests that flow sizes follow a heavy-tailed distribution (see Table 1.1). Consequently, long flows can lead to large queuing delays and thus affect delay-sensitive traffic. While long flows are delay insensitive, they require high throughput because these stem from the need to update internal data structures which can affect the quality of results returned [4], e.g., the *PageRank* algorithm requires updating the ranks of pages by continuously solving a Markov Chain).

Note: Background traffic (doesn't include query traffic) which comprises of short, medium, and long flows follow a *heavy-tailed distribution*. This is a very important point for size-based differentiation because benefits are high with heavy-tailed distribution.

Traffic Type	Size
Query traffic (short flows)	2 KB-20 KB
Delay-sensitive messages (medium flows)	100 KB-1 MB
Throughput-sensitive flows (long flows)	1 MB-100 MB

Table 1.1: Traffic Types and Typical Workloads [1]

Near Real-Time/Soft deadlines

Many web applications are interactive in nature, which means that response time is a key metric. The target response times are typically guided by customer studies after accounting for network latency and other intrinsic delays. These targets typically have soft-deadlines, [1], which in turn translate into deadlines for worker machines. Additional latency affect has a significant impact on operator revenue. For instance, an added latency of 100 ms costs Amazon 1% sales [14].

High capacity, low latency links

Data centers typically have high capacity links, however, RTTs within a data center are only as high as a few 100 us. This suggests that typical BDPs within a data center may not be too large. Note that TCP throughput is related to the buffer size. To maintain high throughput while sustaining low buffer occupancy, the backoff factor should be smaller than in TCP. Consequently, the buffer size as well as the average queue occupancy can significantly contribute to the overall response time.

Knowledge of flow size information

When incorporating flow-size awareness into a congestion control protocol, it is useful to have a-priori knowledge of the flow size. For almost all interactive web applications today, size of flows initiated by aggregators and workers is known in advance [11]. For instance, in case of web search, queries sent by aggregators to the workers are fixed in size, whereas, the responses from workers include the top m matching index records, where m is specified in the query. Consequently, the size of the response is known to the application even before the worker begins processing. The same hold for other services such as key-value stores [15, 16] and data processing [11, 12]. For applications where this condition does not hold, the application designer can typically provide a good estimate of the expected flow sizes. In the absence of such information, a policy, such as Least-Attained Service (LAS), can also be used, which can provide similar performance [17].

1.2 Data Center Transport Requirements

Based on the characteristics of data center environments, any transport protocol designed for data center should consider following design requirements.

High Burst Tolerance

Application workflow patterns often lead to synchronized bursts, therefore, a protocol should be able to provide high burst tolerance to avoid incast.

Those connections are called as barrier synchronized since the final performance is determined by the slowest TCP connection that suffers timeout due to packet losses. The performance collapse of those many-to-one TCP connections is called TCP incast congestion [10].

The traffic and network condition in data center networks create the three preconditions for incast congestion as summarized in [9]. First, data center networks are well structured and layered to achieve high bandwidth and low latency, and the buffer size of ToR (top-of-rack) Ethernet switches is usually small. Second, recent measurement study showed that barrier synchronized many-to-one traffic pattern is common in data center networks [18], mainly caused by MapReduce [11] alike applications in data center. Third, the transmission data volume for such traffic pattern is usually small.

Minimize Flow Completion Times (or Maximize Application Throughput)

The completion time (or response time) of a flow depends on three protocol factors:

- Startup latency,
- Average queue length at the routers, and
- Bandwidth allocation policy.

A protocol should have low startup latency, maintain low queues, and apportion bandwidth among flows as to minimize average flow completion times. In data center the key to reduce average flow completion time is to maintain small queues, because queuing delay can be significant portion of the RTT. Small completion times for small flows/ queries mean increased application throughput.

High Network Utilization

Given enough offered load, a protocol should achieve high network throughput at all times. Achieving high network utilization while ensuring low queue occupancy becomes challenging. Thus, suitable strategy should be adopted to ensure high throughput.

Mechanisms for Speculative Scheduling

Since some flows may have soft deadlines, mechanisms should be in place to pro-actively terminate flows that are likely to miss the deadlines so as to save network bandwidth.

1.3 Transport Mechanism Design Choices

We now discuss the important design choices that help us achieve the goals stated earlier.

Congestion Signal

In order to achieve high burst tolerance, it is important to inform senders about incipient congestion so that they can react early and adjust their sending rates before losses occur.

Congestion control protocols that rely on packet loss as a primary congestion signal find it hard to obtain early congestion information because when packet losses occur router queues are already full. While Active Queue Management (AQM) schemes, such as RED [19] and REM [20], can provide early congestion information, it is challenging for such scheme to simultaneously achieve high throughput and low delay, especially when there is low statistical multiplexing which is common in data center environments. This happens because traditional AQM schemes rely on binary congestion signals.

A multi-bit signal can allow both low delay and high throughput [1]. Delay is a multi-bit signal that can potentially be used for this purpose. However, round-trip times within data centers are typically very small ($\approx 300\,\mu$s), the signal can be quite noisy. Moreover, detecting small changes in delay requires high resolution timers. [1] uses a queue-threshold based AQM scheme to adapt the sending rate based on the fraction of marked packets. While this yields a multi-bit signal, it is coupled with the number of packets sent by a flow. Consequently, flows with smaller congestion window sizes observe higher level of congestion. This can increase the convergence time for newly arriving flows and hence AFCT.

Bandwidth Allocation Policy

To maximize the number of flow completions per unit time, one can use insights from scheduling theory, which informs us that the Shortest Remaining Processing Time (SRPT) [21, 22] scheduling policy minimizes flow completion times in a single-server system [23]. SRPT can provide unbounded gains in response times over the Processor Sharing (PS) policy under heavy load when the flow size distribution is heavy-tailed [24]. SRPT assumes that when a new flow (or a job in the terminology of queuing theory) arrives, the entire flow's packets are present at the bottleneck queue and the router knows the flow-size.

Our approach is to identify practical and close approximations to SRPT, and create a congestion control algorithm that comes close to minimizing FCT. For example, consider the Foreground Background (FB) policy [25], proposed as an approximation to SRPT in the absence of flow size information. FB gives service to the flow with the least attained service at any time instant. If more than one flow has the least attained service, they get served equally. FB policy has a bias towards small flows, and attempts to complete the short jobs as quickly as possible. The upside of FB is it minimizes mean flow completion time (among disciplines which have no knowledge of remaining sizes), for a certain class of heavy-tailed flow-size distributions (such as the Pareto distribution) [26]. The downside is routers still need information on the so-far-completed flow length.

1.4 Thesis Statement

The thesis of this work is that "by introducing flow size awareness, data center traffic characteristics can be leveraged to improve the average flow completion times within data centers".

Achieving such a goal raises several research challenges. The most important one is: If we were to design a transport mechanism for data center, that uses flow size information to apportion bandwidth with flows, from scratch, how would we design it? This challenge leads us to the following questions:

1. How should flow size information be used?
2. What bandwidth sharing mechanism should be used? How does performance of flows of different sizes vary with bandwidth sharing mechanism?

3. Where should bandwidth sharing mechanism be employed, at end-hosts or routers? If employed at routers, how should the capacity share be conveyed to end-hosts?

4. What congestion signal should be used to mitigate congestion and maintain low queue occupancy?

1.5 Thesis Contribution

To address above mentioned questions and challenges, we make following contributions.

We propose a mechanism of sharing network capacity between file transfers of markedly different sizes. We perform this allocation at transport layer. TCP does not differentiate on the basis of file size. We believe using flow size info at transport layer can provide significant advantages. Thus, we leverage heavy-tailed nature of DC traffic size [1]. Below, we present two cases for utilizing flow size information to apportion bandwidth in data centers.

As a first case, we propose an end host based mechanism named as size aware delivery control protocol (SDC), that applies different AIMD rules to a flow based on its residual size. The traffic is divided in various classes, based on its size, which have their own α, β parameters. This way, we prioritize short flows by using larger α, β pair than long flows. We capture protocol performance, using positive systems model, analyticaly. We also implemented SDC in ns-2 to test protocol performance. Detailed description is provided in chapter 3.

Secondly, we propose, router assisted capacity sharing (RACS), a network assisted congestion control protocol. It uses rate feedback from routers to control transmission rate of the flows. Another main difference from SDC is absence of classes. Instead of discretized classes it provide a better aproximation to SRPT, than SDC. We implemented this scheme in ns-2 [27] and compare its performance with RCP and SDC in different DC like settings. Detailed description is provided in chapter 4.

1.6 Thesis Organization

The rest of the work is organized as follows:

Chapter 2 provides a background on data center congestion control protocols and discusses the size-based congestion control related work. Chapter 3, then presents SDC with its performance evaluation and its analytical analysis. This is followed by Chapter 4, which presents the design and simulation study of the RACS protocol. In Chapter 5, we summarize our contributions and discuss limitations of proposed protocols. Finally, we conclude with possible future extensions.

Chapter 2

Related Work

Size based differentiation has been coupled with different TCP versions [24, 28, 29], to improve average flow completion time (AFCT) of the system. Most of these studies have been done analytically with a very little emphasis on implementation in real networks. Below we discuss few such efforts. We also discuss few protocols specifically designed for DC environment and highlight their pros and cons.

2.1 TCP & Size-based Scheduling

Yang et. al [28] proposed TCP SAReno, which adapts AIMD parameters based on the residual flow size. This protocol faces following challenges when used in a DC environment: First, TCP SAReno parameters do not depend on which classes are active in the network. Consequently, long flows can achieve very low throughput even in the absence of short transfers. Second, TCP SAReno builds on TCP Reno and therefore, doesn't address issues like Incast. Finally, since the proposal was a proof of concept, a class-based analysis was conducted.

Zieglar et al. [29] proposed to modulate congestion control parameters to improve the startup latency of short flows. In their scheme, a short flow achieves no higher throughput than that of long flows, in order for the protocol to be incentive-compatible with existing TCP flows. Note that since data centers are under a single administrative control, issues of incentives are much less important than on the Internet. Moreover, like TCP SAReno, it doesn't address issues like Incast.

Balter et al. [24] proposed to improve the performance of web-servers

servicing static HTTP requests by giving priority to short flows by controlling socket buffers. They also studied TCP doing SRPT approximation via processor sharing.

Schrage et al. [23] emphasize the use of scheduling systems in single server systems. Using analytical tools authors shows that SRPT based scheduling can provide unbounded gains over simple PS scheme. Some other scheduling schemes like least attained service (LAS) [17], earliest deadline first (EDF) [30] also highlight the gains introduced in the performance by simple yet intelligent scheduling schemes.

With all the above proposals, long flows can significantly under-utilize the network because they use fixed parameters irrespective of the load and/or the presence of other flows in the network. These works are the basic foundations behind our application of SRPT like scheme in data center networks.

2.2 Data Center Transport Protocols

Recently, keeping in view the challenges imposed by DC, few protocols [1, 9, 10, 31] have been proposed to improve the performance of DC transport. Below, we discuss pros and cons of these schemes.

2.2.1 Incast TCP (ICTCP)

Some application-level solutions to mitigate Incast problems include jittering responses or batching responses in small groups. These solutions mitigate Incast, but increase the median response time. Recently, researchers have shown [9, 10] that lowering the RTOmin to 1ms, and using high-resolution retransmission timers can help alleviate the impact of Incast induced timeouts. But this does not prevent queue buildup; and hence does not address latency issues.

2.2.2 Data Center TCP (DCTCP)

DCTCP [1] achieves low queue occupancy and addresses Incast by trading off the convergence time of new flows which includes latency critical, short flows. This happens because DCTCP couples the backoff factor with the window size of flows. Since the backoff factor is based on the fraction of marked packets, flows with small congestion window sizes, such as new flows, get a larger fraction of their packets marked. DCTP reduces the convergence time

of new flows when competing with existing flows that have a larger window size. The reason is that DCTCP sources compute the backoff factor based on the fraction of marked packets. Since new flows have shorter windows when their packets get marked, they backoff by a greater amount. The authors report that DCTCP can increase convergence times by a factor of 2-3 over TCP. In particular, the AFCT of short flows can be elongated.

2.2.3 D3 Congestion Control

Recently [31] proposed a protocol, named D3, that exploits flow deadlines and size information to assign transmission rate to flows. D3 sets its transmission rate based on the rate feedback from the network. Their main goal is to improve application throughput, which is defined as the number of flows that meet their deadlines. D3 faces two main issues that limit its gains when applied in a real network. First, it does not necessarily reduce flow completion times since deadline-driven flows may not be short. Second, it focuses on hard deadlines that are hard to predict because of queuing delays and congestion in DC network. However, soft deadlines are common in data centers,

Summary

In this chapter, we discussed common approaches followed to aproximate SRPT in computer networks. We also discussed few recents works specific to the DC environment and stated their limitations. In next two chapters, we present our proposed protocols to improve AFCT within DC network.

Chapter 3

Size Aware Delivery Control (SDC)

3.1 A Simple Illustration

Figure (3.1) shows the basic ingredients of our protocol. Traditional TCP schemes assign same level of aggressiveness to all flows. At routers, AQM or ECN based mechanism is employed to control congestion in the network. These schemes give rise to the issues discussed in chapter (1).

Our protocol uses load information at routers to quantify congestion level in the network and adapt congestion window accordingly. Use of load factor allows us to maintain small queues at the routers. Besides, we use different classes to control the behavior of flows with different sizes. This allows us to assign higher throughput to short flows and hence improve application throughput. To make sure long flows do not suffer performance degradation in the absence of short flows, routers provide information about the highest priority flow in the network to end hosts. Benefit is that lower priority class can use parameters of higher class in the absence of higher class, hence improve network throughput.

3.2 Protocol

In this section, we describe different components of the protocol. Some important notations used are listed in Table 3.1:

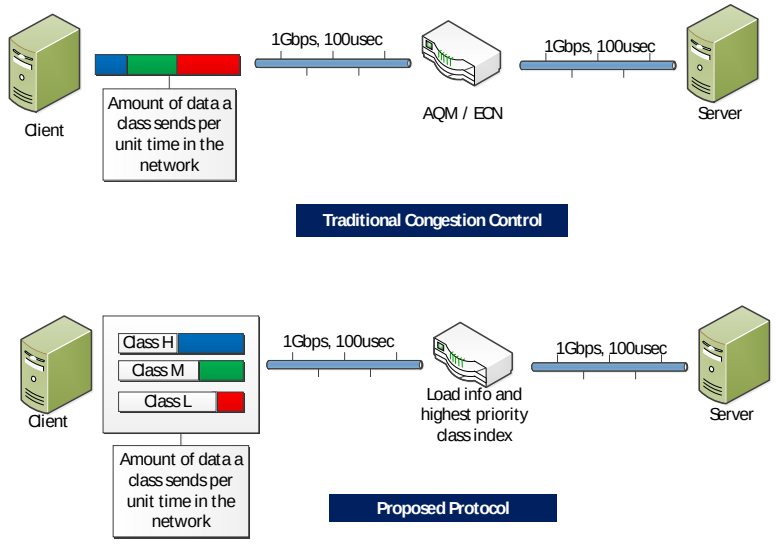

Figure 3.1: Basic Protocol

3.2.1 Sender

The protocol sender maintains three classes: {High, Medium, Low} denoted by A, B, and C, respectively. Each class has an associated $(\gamma_c, \alpha_c, \beta_c, r_c)$ tuple that defines the priority of class $c \in \{A, B, C\}$, where γ_c is the multiplicative increase parameter, α_c the additive increase parameter, β_c the multiplicative decrease parameter, and r_c the maximum remaining flow size belonging to class c when all classes are active.

In particular, we have the following relationship between parameters across different classes

- $(\gamma_A, \alpha_A, \beta_A) \geq (\gamma_B, \alpha_B, \beta_B) \geq (\gamma_C, \alpha_C, \beta_C)$
- $r_A \leq r_B \leq r_C$

where $(u, v, w) \geq (x, y, z)$ means that $u \geq x, v \geq y$, and $w \geq z$.

A flow initially applies multiplicative increase (MI) until the first overload event as

$$w(t + T) = w(t) \times (1 + \gamma_c) \tag{3.1}$$

Protocol	Description
	Sender
γ_c	Multiplicative increase parameter
α_c	Additive increase parameter
β_c	Multiplicative decrease parameter
r_c	Maximum remaining flow size
	Router
f	Load factor on router
C	Link capacity
t_p	Measurement interval
λ	Total bytes arrived in t_p
q_{av}	Average queue maintained at a router

Table 3.1: SDC Parameters

When $f < 1$, a flow applies additive increase (AI) as follows:

$$w(t+T) = w(t) + \alpha_c \tag{3.2}$$

When $f \geq 1$, flows apply multiplicative decrease (MD) to the congestion window size as

$$w(t+\delta t) = \beta_c \times w(t) \tag{3.3}$$

where T is the round-trip time, $\delta t \to 0$, $\gamma_c > 0$, $\alpha_c > 0$, and $0 < \beta_c < 1$.

Flow-to-Class Mapping: Since the goal of protocol is to approximate SRPT at sender, each flow $n \in \{1, 2, .., N\}$ is initially mapped to a class based on its remaining flow size r_n as follows:

- Class A if $r_n \in [1, r_A]$
- Class B if $r_n \in [r_A, r_B]$
- Class C if $r_n \in [r_B, \infty]$

As r_n of flow decreases, they graduate to higher priority classes.

Since the goal of the protocol is to reach a steady state where the system is near full link utilization, this mapping changes dynamically according to the following rules:

- When all classes are active, flows use parameters based on their class.

- When only two classes are active, specifically, B and C, flows use parameters of class A and C, respectively. When classes A and B or A and C are the only classes present, their behavior remains unchanged.
- When only one class is active, it behaves as class A.

Dynamic mapping of flows to classes is useful in ensuring high network throughput at all times. For instance, suppose there are only long flows in the network that start in class C. Since class C uses conservative parameters, this is likely to lead to under-utilization. However, since it is the only class, it should be able to use more aggressive parameters.

3.2.2 Router

The router performs two functions: (a) it computes the f periodically and (b) it maintains the index of the highest class that is currently active. The load factor is computed as follows [32]:

$$f = \frac{\lambda + q_{av}}{Ct_p} \tag{3.4}$$

where t_p is the measurement interval, C is the link capacity, λ is the total bytes arrived in t_p, and q_{av} is the average queue length.

To achive (b), a counter for three classes is maintained at the routers. When a SYN packet of a flow from a particular class is received counter is incremented and is then decremented on receiving FIN packet.

The load factor is conveyed, using ECN bits, to the receiver which sends it to sender using TCP options field. The highest priority class info is communicated using ToS field in IP header.

3.2.3 Receiver

The protocol receiver is the same as that of TCP with the exception that it copies f as well as the index of the highest priority class that is currently active in the acknowledgement packets sent to the sources.

3.3 Design Challenges

Our main goal is to approximate SRPT behavior at End-hosts using minimal information from the network. To achieve so, we use size based bandwidth

allocation policy at end-hosts. This approximation arises many questions that need to be addressed.

How to Approximate SRPT From The End-hosts?

To approximate SRPT at end-hosts we use residual flow size based window adaptation. This is achieved by having (α, β) parameters dependent on remaining size of flow. We distribute flows in class based on their size and assign a (α, β) to that class. It is to be noted that all the flows within a class use same (α, β) parameters. This mechanism gives priority to short flows and helps us improve AFCT.

What (α, β) Parameters to Choose for Each Class?

α and β play a major role in avoiding Incast and achieving high throughput. In data center networks we have very small BDPs, thus α, β needs to be controlled intelligently.

1. For α we may not have too much room above 1, as it would increase burstiness and create Incast problem. Which as a consequence may also result in increased AFCT of short flows. Therefore, we have $\alpha < 1$ in our case.

2. For β, we have a wide range of options that can be used. We can vary it from 0.1 to 0.9 depending on the network load at bottleneck link and flow size.

SRPT Approximation: Can Classes be Avoided?

We use SRPT for flow size based control laws, i.e., adapt our control laws so that long flows are less aggressive than short flows. One way to achieve it is by having control laws as a function of flow size as done in [29, 28]. In DC environment we believe this can reduce throughput of a long flow. As a worst case scenario if we have only one long flow, as we maintain small buffer occupancy and use less aggressive parameters (control laws), we may not use link fully.

To achieve full link utilization: a) A long flow may use aggressive parameters (same as that of short flow) or b) Another approach can be to dynamically adapt size of buffers at routers, such that in the presence of

long flows only we have higher buffer occupancy, otherwise low buffer occupancy.

To achieve, a), the end hosts must have information regarding current highest priority flow in the network. Using b), we can achieve higher throughput for long flows as well. But it has a downside that latency of short flows increases. We need to identify queue thresholds and which packets to be dropped when a short flow joins the network. As DC traffic is largely comprised of short flows so we may also have frequent oscillations in buffer size requirements at routers. Therefore, it is necessary to have classes to reduce burden of such signaling mechanisms.

3.4 Analysis

In this section, we analyze the impact of different parameters on the performance of SDC using a model. In particular, we study how the choice of class parameters allows us to achieve different performance tradeoffs in terms of throughput and AFCT.

3.4.1 Model

Consider N flows, indexed by $i = 1, \ldots, N$, sharing a single bottleneck link. We model this as an AIMD system in which each flow can use different α_i and β_i parameters, similarly to the models of [33, 34].

Let $R_+ = [0, \infty)$ and let $(\cdot)'$ denote transpose. The state of the system is a vector $w(k) \in R_+^N$ of window sizes indexed by a discrete time variable k. All amounts of data will be measured in units of packets (assumed to be equal for all flows) and all times will be measured in units of the slot, t_p.

Flows are assumed to be synchronized and for ease of presentation, we ignore the connection setup time, which can be easily incorporated into the model. We assume that a flow spends most of its time in the AIMD phase, which is true when the flow size is large relative to the per-flow BDP of the path; which largely holds in data center environments [1].

We assume flows to have homogeneous RTTs equal to $T = t_p$ and that each source is informed of congestion one RTT after the overload event. Define a backoff event to be a slot in which f exceeds 100%. We model the

total traffic arrived as

$$\sum_{i=1}^{N} w_i(k) = P + \sum_{i=1}^{N} \alpha_i, \forall k > 0 \tag{3.5}$$

where $w_i(k) \geq 0, \forall i, k \geq 0$, $P = C_l T_d$ is the maximum number of packets that can be in transit in the network at any time, C_l is the link capacity in packets per second and T_d is the round-trip time when the queue is empty.

At the $(k+1)$th backoff event, we have

$$w_i(k+1) = \beta_i w_i(k) + \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j} \left[\sum_{i=1}^{N} (1-\beta_i) w_i(k) \right]. \tag{3.6}$$

Then the dynamics of the entire network of sources are described by [33]

$$W(k+1) = A(k)W(k), \tag{3.7}$$

where

$$A(k) = diag(\beta) + \frac{1}{\sum_{j=1}^{N} \alpha_i} \alpha(e' - \beta'), \tag{3.8}$$

$W(k) = [w_1(k), \ldots, w_N(k)]$, $\beta(k) = [\beta_1, \ldots, \beta_N]$, $\alpha = [\alpha_1, \ldots, \alpha_N]$, and $e = (1, \ldots, 1)' \in R^{N \times 1}$.

3.4.2 Average Throughput

We now use the above model to determine the average throughput of flows when AIMD sources using α_i and β_i parameters co-exist. By [33], the network described by the model (3.5)−(3.8) converges to a unique stationary point $W_{ss} = \theta x_p$, where x_p is

$$x_p^T = \left[\frac{\alpha_1}{1-\beta_1}, \frac{\alpha_2}{1-\beta_2}, \ldots, \frac{\alpha_N}{1-\beta_N} \right] \tag{3.9}$$

and θ is a positive constant such that constraint (3.5) is satisfied. Therefore,

$$\theta = \frac{P + \sum_{i=1}^{N} \alpha_i}{\sum_{i=1}^{N} \frac{\alpha_i}{1-\beta_i}} \tag{3.10}$$

Note that the vector W_{ss} provides the maximum congestion window of flows i.e. the window size at a backoff event once the network has converged. Therefore, the maximum throughput $S_{i,max}$ of a flow i is then given by:

$$S_{i,max} = \frac{\theta \cdot \alpha_i}{T(1-\beta_i)}. \tag{3.11}$$

In steady state, the window size of each flow varies between $\frac{\beta_i \theta \alpha_i}{1-\beta_i}$ and $\frac{\theta \alpha_i}{1-\beta_i}$. Consequently, the long-term average throughput of flow i is given by

$$S_{i,av} = \frac{\theta \cdot \alpha_i}{T(1-\beta_i)} \left(\frac{1+\beta_i}{2} \right). \tag{3.12}$$

The average network throughput is then given by:

$$S = \frac{1}{T} W_{ss} \beta'_{av} \tag{3.13}$$

where $\beta_{av} = \frac{1}{2}[1+\beta_1, 1+\beta_2, \ldots, 1+\beta_N]$.

If a flow spends most of its time in the AIMD phase, then the flow completion time can be approximated by

$$T = d/S_{i,av} \tag{3.14}$$

where d is the flow size in packets.

3.4.3 Application Throughput

An important metric that most operators are interested in is number of flows completed per unit time in their network. It is important to compare the performance of proposed protocol under different class settings. Let's say T is the reference period to compare different protocol settings. Then, for c number of classes we can estimate the total number of flows completed as follows:

$$F_{total} = \sum_{i=1}^{c} \frac{TN_i}{AFCT_i} \tag{3.15}$$

.

Where N_i is the number of flows completed in class c.

3.4.4 Approximating SRPT

The analysis in the previous section allows us to determine the throughput and AFCT of AIMD sources using different but fixed parameters. This can be viewed as a Weighted Processor Sharing (WPS) system where each queue has a priority defined by the AIMD parameters. However, with SDC, which aims at approximating SRPT, a flow may traverse one or more classes during its transfer and hence may change its parameters as a function of the remaining flow size.

To capture this behavior, we change α_i and β_i parameters of each source depending on its size as well as the class thresholds, which together impact the time a flow spends in each class. For instance, a flow which sends most of its packets in class C uses class C parameters for a large fraction of its packets. Therefore, for the jth class, where $j \in \{2, .., c\}$ we have

$$\alpha'_j = \alpha_j \cdot \frac{d_j - d_{(j-1)T}}{d_j} + \sum_{i=1}^{j-1} \alpha_i \cdot \frac{d_{iT} - d_{(i-1)T}}{d_j}, \tag{3.16}$$

$$\beta'_j = \beta_j \cdot \frac{d_j - d_{(j-1)T}}{d_j} + \sum_{i=1}^{j-1} \beta_i \cdot \frac{d_{iT} - d_{(i-1)T}}{d_j} \tag{3.17}$$

where d_j is the size of a flow starting in class j, d_{jT} is the class threshold/maximum size of class j, and $d_0 = d_{0T} = 0$. Since $j = 1$ refers to the highest priority class, the parameters used by flows starting in this class do not vary over their lifetime.

Example. As an example, consider the following scenario: Suppose a single flow with size 1000 packets starts in class C. Let the thresholds of class A and B be 50 and 100 packets, respectively. Then a flow employs the following average AIMD parameters:

$$\alpha' = 0.9\alpha_C + 0.05\alpha_B + 0.05\alpha_A$$
$$\beta' = 0.9\beta_C + 0.05\beta_B + 0.05\beta_A$$

The above analysis shows that the performance of the network can entirely be controlled by the operator based on the suitable choice of class sizes and α, β values of that class.

3.4.5 Class Based Analysis

In this section, we perform analysis based on the model presented in the previous section to help guide the choice of key protocol parameters. To aid

Class A	Class B	Class C
G_{AC}=2	G_{BC}=2	(0.5, 0.5)
G_{AC}=5	G_{BC}=2	(0.5, 0.5)
G_{AC}=10	G_{BC}=2	(0.5, 0.5)
G_{AC}=50	G_{BC}=2	(0.5, 0.5)
G_{AC}=100	G_{BC}=2	(0.5, 0.5)

Table 3.2: Varying Class A Gain

analysis, we define the throughput gain of one class over the other as,

$$Gain = G_{12} = S_1/S_2 = \frac{\alpha'_1(1-\beta'_2)(1+\beta'_1)}{\alpha'_2(1-\beta'_1)(1+\beta'_2)} \tag{3.18}$$

where S_j, α'_j, and β'_j, $j \in \{1, 2\}$ is the throughput, additive increase, and multiplicative decrease parameters for flow j, respectively. This ratio also helps us estimate α, β tupple for one class based on the knowledge of gain and α, β tuple of another class.

Here, for simplicity of analysis we assume that load in the network is constant. For ease of tractability, we use three classes (A, B and C) to approximate the behavior of SRPT. While more classes can allow more closer approximation, it can complicate the choice of parameters. Note that the model is applicable to any number of classes. The class thresholds are taken as $da = 50$, $db = 100$ and $dc = 100000$ packets. These thresholds are based on flow sizes observed in real data centers [1, 4]. These can be varied in accordance with the observed flow patterns. Unless stated otherwise, in each of the classes, we set number of flows as $Na = 15$, $Nb = 5$, $Nc = 1$ to reflect a heavy-tail distribution [1], and $RTT = 300\mu$s, which reflects a typical RTT within a data center.

Impact of the Degree of Priority on Performance

In this section, we analyze the tradeoff between the improvement in AFCT of higher priority traffic and the throughput degradation of low priority traffic. We capture this tradeoff by adjusting the throughput gain G_{ij}. We pick one class, vary its gain over other classes and analyze the impact on AFCT and throughput of different classes.

In the first scenario, we vary the gain of class A with respect to class C while using fixed parameters for class C and class B (see Table 3.2). Figure

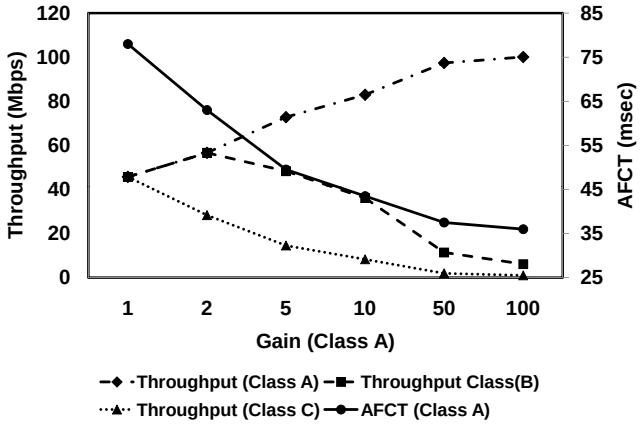

Figure 3.2: Network Behavior for Class A Gain

(3.2) shows the AFCT of class A flows and the throughput of class B and class C flows as function of G_{AC}. Observe that as the gain of class A increases from 2 to 100, its AFCT improves by ≈30 ms, whereas, the throughput of class B and C reduces to ≈5 Mbps and 1 Mbps, respectively. The results also show that beyond a gain of 10, we get diminishing returns for class A at the cost of a significant throughput degradation for class B and C. Gain setting of 1 shows that all classes get fair share of the bandwidth when using same parameters.

Next, we set G_{AC} =10 based on the observations from the previous scenario and vary G_{BC}. Table (3.3) shows the corresponding settings. Figure (3.3) shows that as the gain of class B increases, its AFCT improves significantly. This happens because with the increase in gain, the aggressiveness of class B approaches that of class A. We can see that throughput of class B increases at the cost of drop in gain of class A. This behavior is dependent on the effective α, β values that it gets depending on the class thresholds and gain over other classes. Thus, it is important to analyze the effect of class thresholds on the algorithm performance, which is the subject of next section.

Class A	Class B	Class C
G_{AC}=10	G_{BC}=1	(0.5, 0.5)
G_{AC}=10	G_{BC}=2	(0.5, 0.5)
G_{AC}=10	G_{BC}=5	(0.5, 0.5)
G_{AC}=10	G_{BC}=10	(0.5, 0.5)

Table 3.3: Varying Class B Gain.

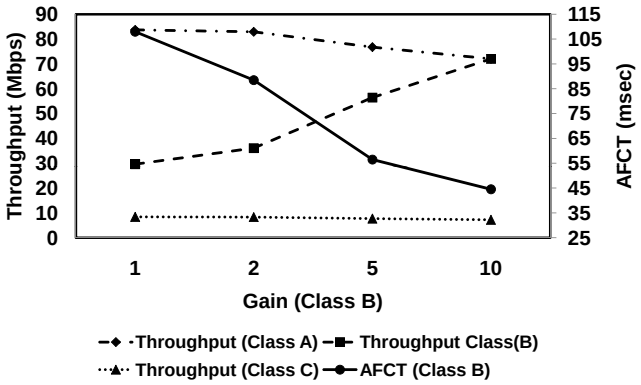

Figure 3.3: Network Behavior for Class B Gain

Impact of Class Thresholds

The AFCT of a flow depends on the time a flow spends in each class, which in turn depends on the throughput it achieves in each class. This is closely tied with the size of a flow as well as the size threshold for the class. As shown above, in class based system the performance of a flow is determined by the share of the bandwidth it gets during its lifetime. To capture this effect, we set $db = 100$, $dc = 1000000$ packets and vary da to show the effect on throughput of class A and B.

Figure (3.4) shows that as the class threshold of class A increases, its throughput decreases. But, we get significant gains in throughput of class B. This holds true for flow size in the range of class B threshold only. In addition, class C get diminishing performance improvements. One of the

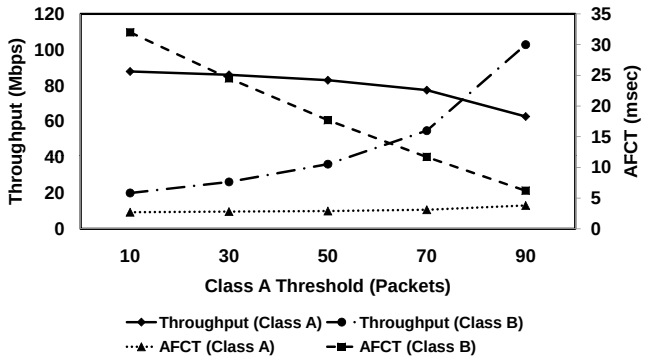

Figure 3.4: Impact of Class A Threshold

key take away from this is that, if we have too many classes and they do not have significant difference in gains, we get minimal advantage in performance. Thus, having too many classes is not optimal from protocol management perspective. This happens because the effective α, β parameters used in a class depend on their size thresholds. And if there are many classes without much difference in their gains, we have to control many variables at the cost of minimal performance improvement.

Next, we study the impact of varying the class A threshold on the β parameter[1]. Figure (3.5) shows that when the class A threshold increases, sources starting in class B become more aggressive and as a result experience throughput increase. But as the total number of flows increases, small flows share bandwidth with more flows, hence throughput of small flows starts degrading.

Above analysis, shows that our model captures the SRPT behavior effectively. Variation in class threshold here captures the bandwidth share a class gets, which indirectly captures the time a flow spend in each class.

[1]Note that the effective α does not change since both class A and class B use the same values.

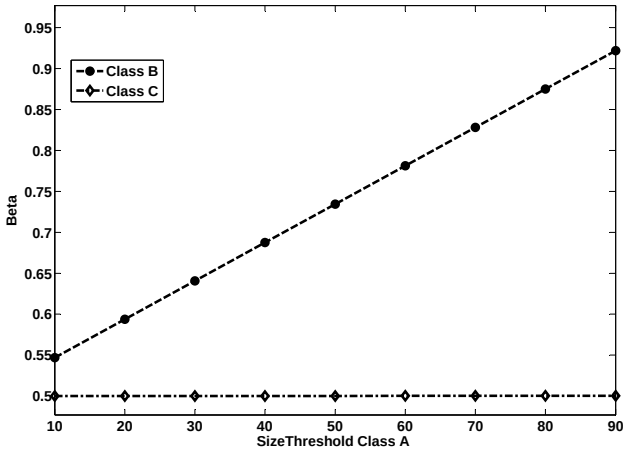

Figure 3.5: Class A Threshold vs Beta of Class B and C

3.5 Evaluation

In this chapter, we evaluate and compare the performance of Size Aware Delivery Control Protocol (SDC) with TCP SACK and TCP SACK+RED/ECN in DC environment. The main goal is to check working of SDC as a congestion control protcol and then test its performance based on the insights from Section (3.4) in chapter (3).

First, we evaluate the basic properties of the SDC algorithm, as a congestion control protcol. We examine its throughput and queuing behavior in single and multi-hop environment. Second, we show a series of micro-benchmarks that explain how SDC ameliorates the data center specific performance impairments, such as Incast and large queue build-ups and provides us benefits in AFCT. Finally, we evaluate SDC using a benchmark generated from the traffic measurements in [1, 31].

We use the topology shown in Figure 3.6 with $1\,Gbps, 300\,\mu sec$ links. We use static buffer size of 100 packets at the routers as in [1], unless otherwise mentioned. Here only three servers in each rack are shown but in some scenarios, we assume that there are more than three servers within a Rack.

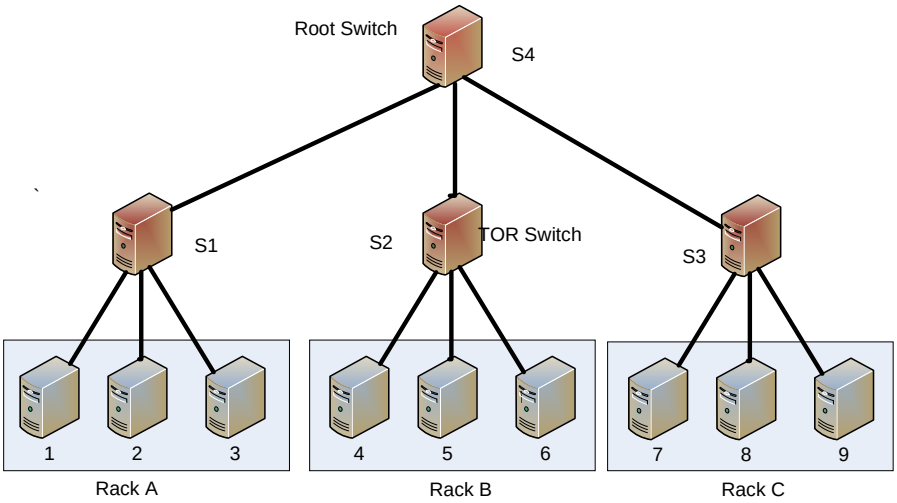

Figure 3.6: Network Diagram

3.5.1 Basic Testing

Our main goal here is to test whether SDC achieves the same throughput as TCP on long-lived flows. Below we discuss two scenarios.

Single Bottleneck Scenario

In this scenario, we assume multiple hosts are connected within one Rack with ToR switch S_1, Figure 3.6. One host acts as a receiver and all other hosts send data to this machine, through ToR Switch (S_1). Long-lived flows are started between senders and the receiver. We repeat experiment with different number of senders sending data to the receiver.

Figure 3.7 shows that SDC maintains at least 95% link utilization. This behavior remains same as the number of senders is increased. This shows that with SDC, long flows can maintain high throughput in the absence of short flows while maintaining small queues (see Figure 3.8),

On the other hand, SACK-Droptail achieves about ∼3% higher throughput but maintains large average queue length as shown in Figure 3.8. While

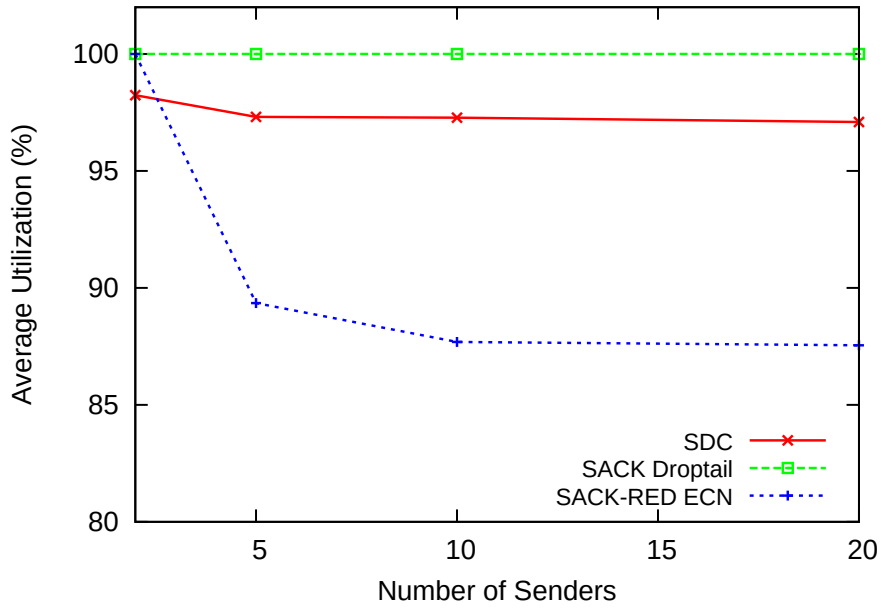

Figure 3.7: Long Flows Only: Average Throughput

SACK+RED/ECN maintains smaller queues, its average throughput degrades as the number of senders is increased.

Multi-hop Networks

To evaluate SDC performance in a multi-hop, multi-bottleneck environment, we use a three hop topology, using Rack A, S_1, S_4, S_2 and Rack B, Figure 3.6. A total of 4 flows, $1 \rightarrow 2, 2 \rightarrow 3, 4 \rightarrow 3$ and $4 \rightarrow 5$are generated. First three flows share a bottleneck switch S_1, and the last flow shares link from host 4 to S_2. In this multi-bottleneck, multi-hop scenario, first three flows achieve an average throughput of 298 Mbps each, and last flow an average throughput of 604 Mbps. This shows that SDC protocol performs well in bottleneck scenarios.

3.5.2 Data Center Specific Impairments

In this section, we compare the performance of protocols under typical data center like environment. First, we consider the Incast scenario. Second, we consider the case when only short queries and long flows co-exist. This scenario help us capture the improvements, SDC brings, in terms of AFCT,

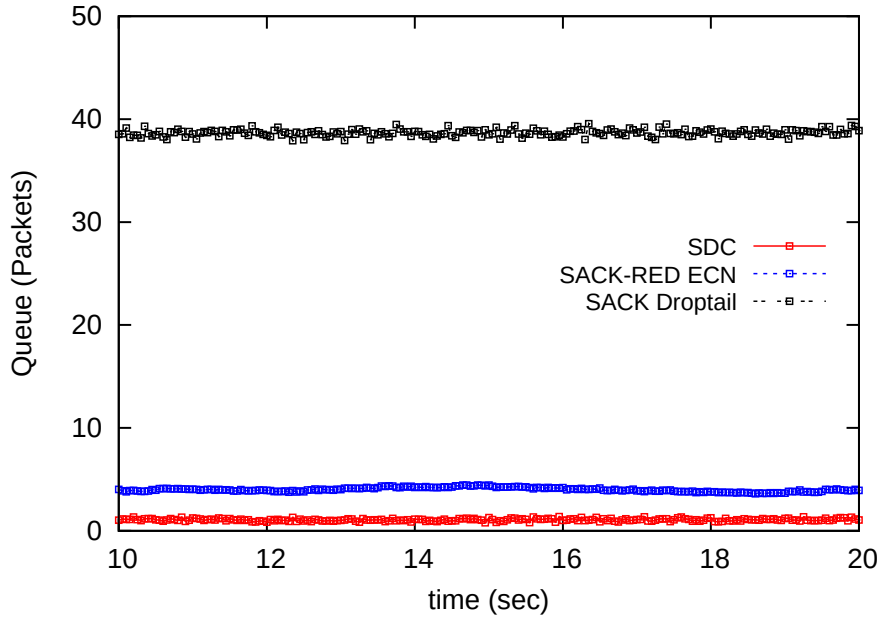

Figure 3.8: Long Flows Only: Queue Length

small queue build-ups and high link utilization.

Incast Scenario

In SDC, we make short flows more aggressive to increase their network share of bandwidth. It may cause Incast problems in small BDP environments like DC. Thus, to predict SDC behavior in sudden flash crowd scenario, we use topology similar to the long lived flows scenario. Each client requests 25 KB queries from n different servers, and each server responds with the requested amount of data. This pattern is repeated 500 times.

Figure 3.9 shows the AFCT of different protocols in this scenario. We can see that using RTOmin $\approx 10msec$, helps TCP mitigate Incast, as shown by very small few losses in Figure(3.10). Observe that all protocols achieve roughly similar AFCT performance when the number of servers is small. However, as the number of servers increase SDC shows improvement in AFCT. With TCP, loss rate increases with the number of servers as shown in Figure 3.10.

While, this is not the case with SDC, it avoids incast by having zero losses

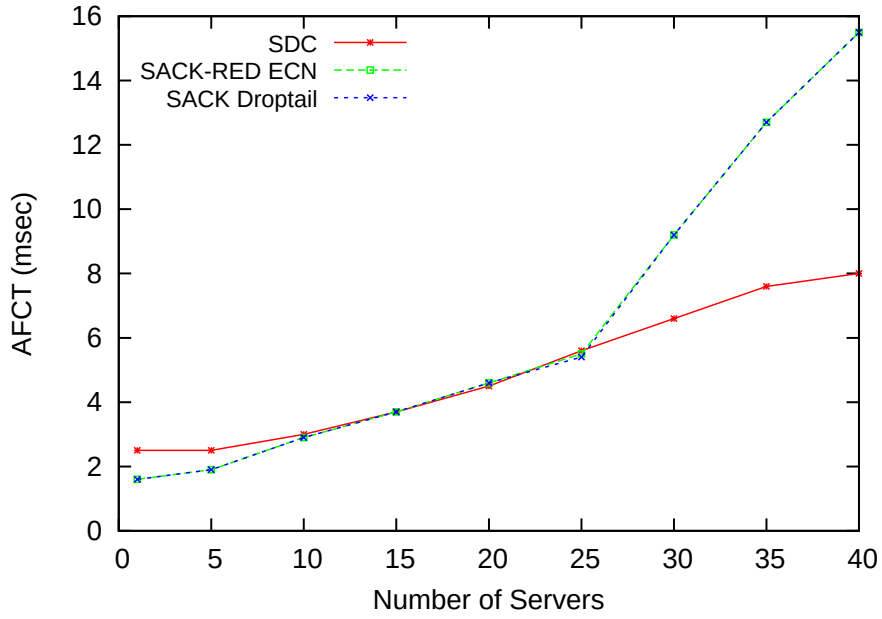

Figure 3.9: Incast: AFCT Comparison

and also maintains small queues, which results in overall AFCT improvement.

Mix of Short and Long Flows

Performance analysis of SDC in a scenario, where flows from Class A and Class C co-exist in the network is of significant importance. In such a scenario, long flows may occupy buffer space and as a result performance of short queries degrades. To test this scenario we connect four hosts to a switch S_3, Figure3.6. Two hosts generate long-lived background traffic. One host acts as a receiver and one host generates the response flow lengths uniformly distributed across [2KB, 50KB]. This, represents the 75th percentile traffic multiplexing in data center networks [1].

Figure (3.11), shows the comparison of SDC and SACK under above conditions. All the protocols maintain high throughput TCP $\approx$ 99% and SDC $\approx$ 98%, but there is a significant difference in the AFCT of flows across protocols. There are three reasons for this: First, SDC assigns higher priority to short flows, maintains small queues (see Figure 3.12), and introduces negligible loss rate. On the other hand, TCP SACK maintains large queues, which considerably increases the latency of short flows.

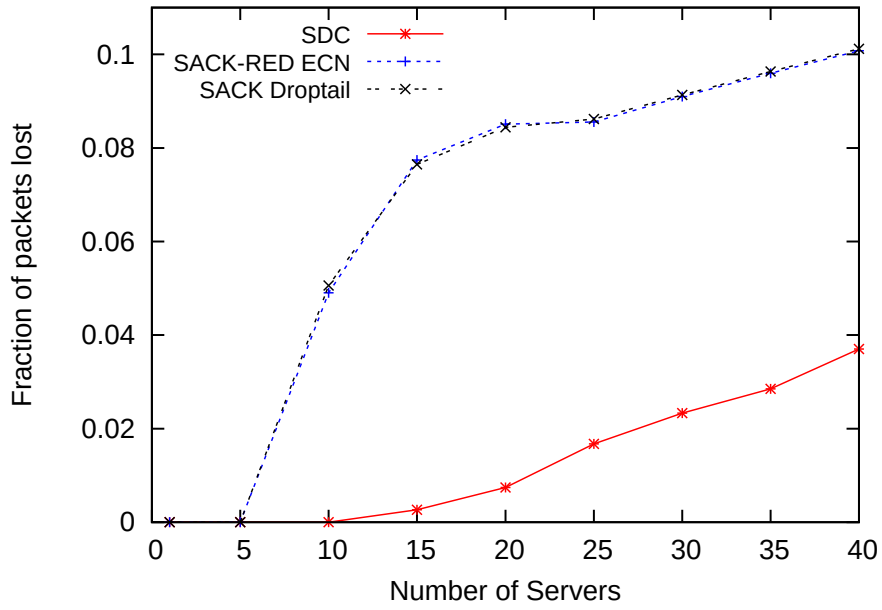

Figure 3.10: Incast: Loss Behavior

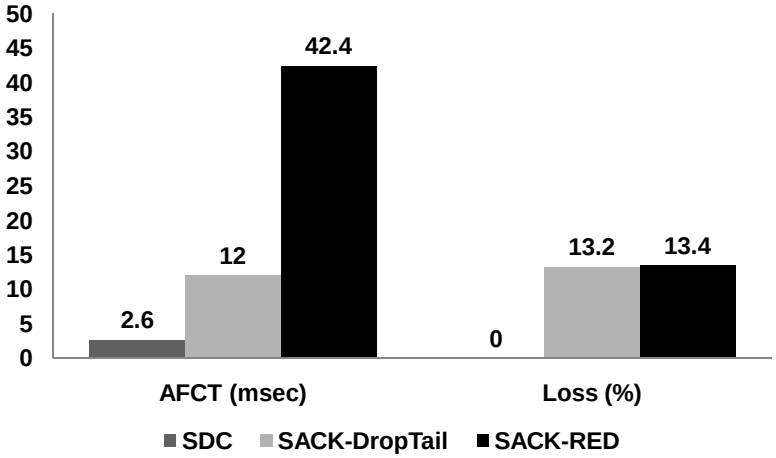

Figure 3.11: Mix Flows: AFCT and Loss Rate

	SDC	SACK	SACK w ECN
AFCT (msec)	2.6	12	42.4
Utilization(%)	98.11	99.99	99.99
Loss Rate	0	0.134	0.132

Table 3.4: SDC Comparison with TCP

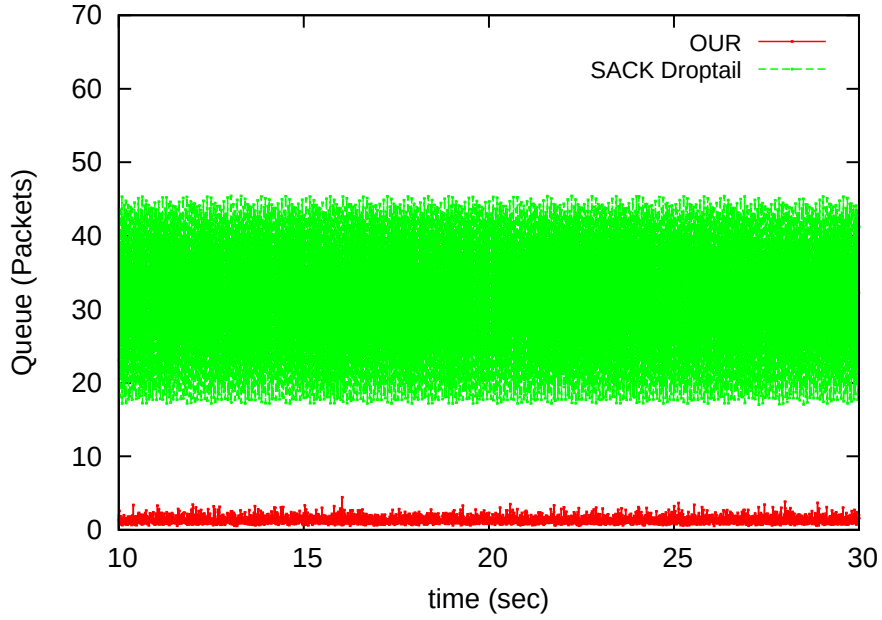

Figure 3.12: Mix Flows: Queue Size (Packets)

3.5.3 Benchmark Traffic

In case of mix of short and long flows we observed that SDC improves AFCT performance significantly. In this section we discuss SDC performance under more general settings common in most data center networks. We generate Pareto distributed traffic with shape parameter 1.2 and average response size of 25KB. To have all mixes of flows we generate two long flows in the background. Figure 3.13 shows that three protocols show comparable behavior at low loads, but as the arrival rate of flow increases, AFCT of TCP flows increases significantly, largely because of large queue buildups and increased loss rate. On the other hand, performance of SDC largely remains same even at arrival rate of 2000 flows/sec.

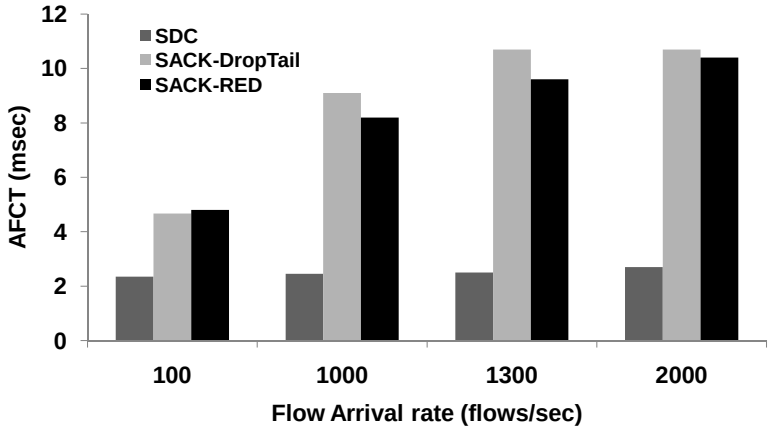

Figure 3.13: Benchmark Traffic: AFCT

3.6 Summary

In this chapter, we proposed SDC, with a design goal, to improve AFCT in data center. Through analysis and ns-2 simulations we show that SDC provides significant advantages when compared to TCP SACK w/o ECN. SDC is a class based protocol, which limits its capacity to approximate SRPT. So we ask ourselves a question, is it possible to approximate SRPT in a better way? Can we do it without classes? This resulted in RACS, that we discuss in next chapter!

Chapter 4

Router Assisted Capacity Sharing (RACS)

Figure (4.1) shows the basic ingredients of routed assisted capacity sharing (RACS) protocol. RACS needs applications to share flow size information. The source endhost uses this information to compute the weights to control the behavior of flows with different sizes. The weight w_t is computed as follows:

$$w_t = w_{min} + w_{max} * \exp(-af_r(t)) \tag{4.1}$$

where w_{min}, w_{max} and a are system parameters and f_r is the remaining flow size. Routers quantify share of capacity based on this weight and inform sources about transmission rate.

4.1 Protocol

Next, we describe different components of the protocol. Some important notations used are listed in Table 4.1:

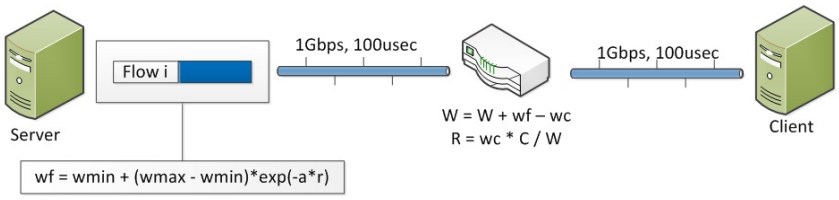

Figure 4.1: Basic Protocol

Protocol	Description
	Sender
w_p	Weight based on previous remaining size
w_c	Weight based on current remaining size
t_{trans}	Length of transmission round
	Router
R	Rate assigned to a flow
C	Link capacity
t_p	Measurement interval

Table 4.1: RACS Parameters

4.1.1 Sender

The protocol sender maintains flow weight information, that it shares with the network, once every transmission round, t_{trans}. Sender estimates, using equation 4.1, the weights w_c, based upon remaining size of the flow.

Sender updates routers with its weight by sending a rate request packet, RR, every t_{trans}. A RR packet, shown in figure 4.2, carries w_c, w_p, $Rate$, RTT, & t_trans fields. Each flow starts by sending a SYNC packet with $w_p = 0$ and w_c calculated based on flows' initial size. In next round a packet with rate request is sent with $w_p = w_c$. In every round w_p & w_c are updated. At the end of transmission, sender sends a FIN packet with RR as , $w_c = 0$. Transmission round length, t_{trans}, is adjusted based on feedback from the receiver, which is the smallest measurement interval (t_p) across the path. Source gets a feedback from the network about transmission rate, calculated using equation 4.2 according to its weight. The source sends data at allocated rate for a time t_{trans} while piggybacking a rate request on one of the data packets.

4.1.2 Receiver

The protocol receiver is the same as that of TCP with the exception that it copies R and t_{trans} in the acknowledgement packets sent to the sources.

Figure 4.2: Rate Request Format

4.1.3 Router

For a link with capacity C, the rate R allocated to i^{th} flow is given as follows:

$$R_i = \frac{w_i * C}{W} \tag{4.2}$$

where w_i is the weight of flow and W is the sum of weights of flows traversing that link. The rate allocated to a flow is valid for the next $t_{trans} \approx RTT$, after which the flow must request again.

Thus, a rate request at time t serves two purposes: (1). It requires the router to update W estimate, and (2). It requires router to allocate rate $R(t)$ for the next transmission round.

4.2 Design Challenges

The foregoing design raises some questions, which are now addressed.

Slotted rate allocation

The rate allocation description above assumes the router has the rate requests for all flows at the same point in time. In reality, the router needs to make allocation decisions in an online, dynamic setting, i.e., rate requests are spread over time, and flows start and finish.

Performing both the actions in a single slot is not possible. Thus, the rate allocation operates in a slotted fashion (from the perspective of the end-hosts). We call this slot, t_p, measurement interval. In one time slot router

calculates W based on the w_c of all the flows passing through it. In next slot it assigns rate to the flow based on this estimate.

To achieve (1), the router needs to know the weight of the flow, which is conveyed through probe packets. To achieve (2), the router must know the previous weight which was used to estimate W, otherwise capacity estimate will be wrong. For this, a router could maintain weights of each active flow passing through it. However, since most flows are very short, such an approach is too heavy-weight, not to mention router memory intensive. We avoid the need for per-flow state on routers by relying on endhosts to convey w_p of each flow. Specifically, each rate request packet, apart from the future weight w_c, contains the weight, w_p, used to estimate W in the previous interval.

Given this, we can now describe how packets are processed by routers. Packets without rate request headers are forwarded just as today. The router first uses the packet header information to perform bookkeeping. The router performs two functions: (a) it computes the W and $avgRTT$ periodically and (b) For each RR, it copies rate $R(t)$ and t_p, if its estimate is lower than current values in packet header.

For every incoming RR, W is updated as follows:

$$W = W + w_c - w_p \tag{4.3}$$

Note: Router sets $t_p = avgRTT$. At sender we have round length, t_{trans}, which is the minimum of this value along the flow path.

Incast & initial rate assignment

Bursts of flows are common in data centers. Such bursts are particularly challenging because of tiny RTTs in the data center. With a typical RTT of $300\mu\,s$, a new flow sending even just one 1500-byte packet per RTT equates to a send rate of 40Mbps! Most TCP implementations shipping today start with a send window of two packets and hence, a mere 12-13 new flows can cause queuing and packet loss on a 1Gbps link.

With RACS, a new flow starts with a rate request packet with the SYN flag set. This adds the flow in the network and updates W. Next we send a rate request packet which gets the rate allocation for that flow. Thus, a flow is first admitted in the network and then assigned a transmission rate. This

implies that a new flow does not make progress for an extra $t_{trans} \approx RTT$ at startup. But, RTTs are minimal, and this approach trades-off a minor overhead in bandwidth and latency (one RTT) for burst tolerance. Our evaluation shows that this vastly improves RACS's ability to cope with flow bursts over the state of the art. This is the case only if router buffers are allocated in bytes. If router buffers are calculated in terms of packets then we do not see much of a gain.

Rate probing

In above rate assignment policy, it is also possible that a flow might get rate of less than one packet per t_{trans}. In this case we can either reduce packet size or send packets less frequently. To ensure a flow do not starve completely, we use second approach, flows are assigned a base rate that allows them to send a header-only packet per t_{trans}. Our protocol also has the flexibility to adjust rate in terms of packets per t_{trans}, which help avoiding starvation of long flows.

4.3 Evaluation

In this section, we evaluate and compare the performance of Router Assisted Capacity Sharing (RACS) with RCP and SDC in DC environment. We use same simulation topologies and scenarios for performance evaluation as chapter 3.

4.3.1 Basic Testing

Our main goal here is to test whether RACS achieves the same throughput as RCP & TCP on long-lived flows. Below we discuss two scenarios.

Single Bottleneck Scenario

In this scenario, we assume multiple hosts are connected within one Rack with ToR switch S_1, Figure 4.3. One host acts as a receiver and all other hosts send data to this machine, through ToR Switch (S_1). Long-lived flows are started between senders and the receiver. We repeat experiment with different number of senders sending data to the receiver.

Figure 4.4 shows that RACS maintains 100% link utilization. This behavior remains same as the number of senders is increased. This shows that

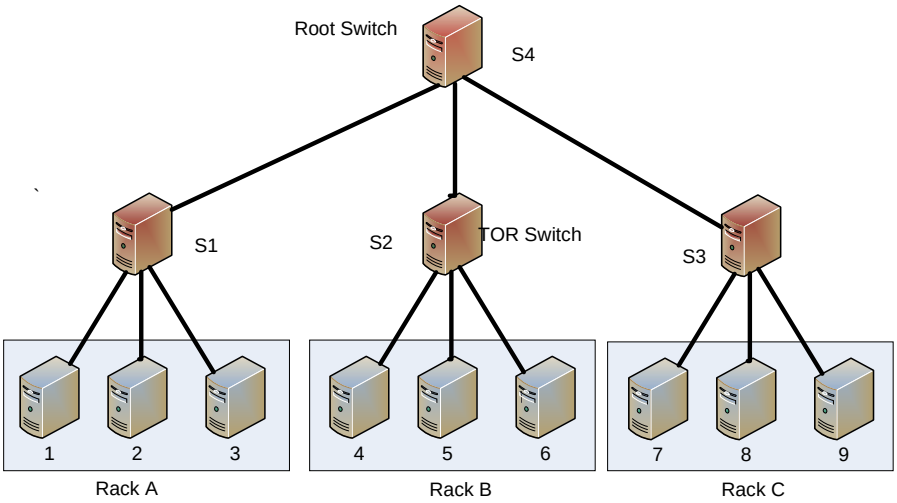

Figure 4.3: Network Diagram

with RACS, long flows can maintain high throughput in the absence of short flows while maintaining small queues (see Figure 4.5). On the other hand, throughput of RCP is low for small number of flows as RCP follows a conservative approach while updating window size.

Multi-hop networks

To evaluate RACS performance in a multi-hop, multi-bottleneck environment, we use a three hop topology, using Rack A, S_1, S_4, S_2 and Rack B, Figure 4.3. A total of 4 flows, $1 \rightarrow 2, 2 \rightarrow 3, 4 \rightarrow 3$ and $4 \rightarrow 5$are generated. First three flows share a bottleneck switch S_1, and the last flow shares link from host 4 to S_2. In this multi-bottleneck, multi-hop scenario, first three flows achieve an average throughput of 300 Mbps each, and last flow an average throughput of 600 Mbps. This shows that RACS protocol performs well in bottleneck scenarios.

4.3.2 Data Center Specific Impairments

In this section, we compare the performance of protocols under typical data center like environment. First, we consider the Incast scenario. Second,

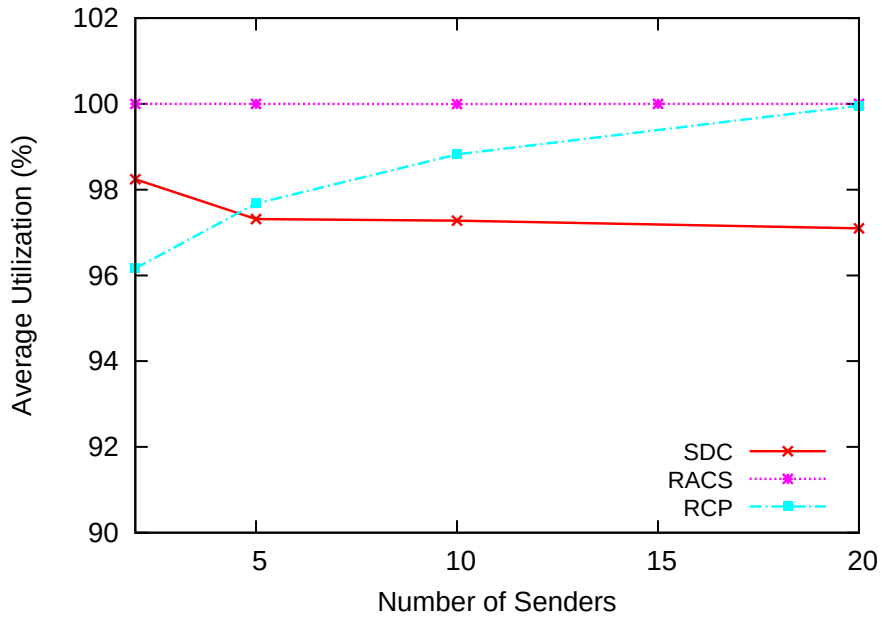

Figure 4.4: Long Flows Only: Average Throughput

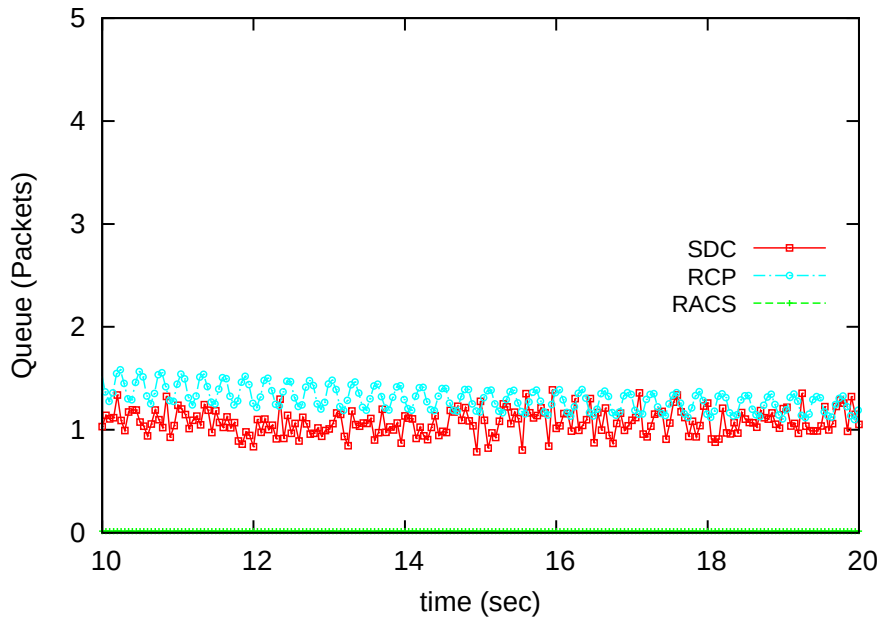

Figure 4.5: Long Flows Only: Queue Length

we consider the case when only short queries and long flows co-exist. This scenario help us capture the improvements, RACS brings, in terms of AFCT, small queue build-ups and high link utilization.

Incast Scenario

In RACS, we make short flows more aggressive to increase their network share of bandwidth. It may cause Incast problems in small BDP environments like DC. Thus, to predict RACS behavior in sudden flash crowd scenario, we use topology similar to the long lived flows scenario. Each client requests 25 KB queries from n different servers, and each server responds with the requested amount of data. This pattern is repeated 500 times.

Figure (4.6) shows the AFCT of different protocols in this scenario. Observe that SDC & RACS achieve roughly similar AFCT performance. However, as the number of servers increase RACS shows improvement in loss rate, Figure(4.7). With SDC & RCP, loss rate increases with the number of servers as shown in Figure 4.7. While, this is not the case with RACS, it avoids incast by having zero losses and also maintains small queues, which results in overall AFCT improvement. Reason being the slotted rate update mechanism of RACS.

Mix of Short and long flows

Performance analysis of RACS in a scenario, where short and long flows co-exist in the network is of significant importance. In such a scenario, long flows may occupy buffer space and as a result performance of short queries degrades. To test this scenario we connect four hosts to a switch S_3, Figure4.3. Two hosts generate long-lived background traffic. One host acts as a receiver and one host generates the response flow lengths uniformly distributed across [2KB, 50KB]. This, represents the 75th percentile traffic multiplexing in data center networks [1].

Figure (4.8), shows the comparison of RACS, RCP and SDC under above conditions. All the protocols maintain high throughput, as shown in Table (4.2), but there is a difference in the AFCT of flows across protocols. We can see, Figure (4.8), that RACS outperforms RCP & SDC in terms of AFCT. This gain can be increased by increasing the gain of short flows over long flows. Also, all the protocols maintain small queues, Figure (4.9).

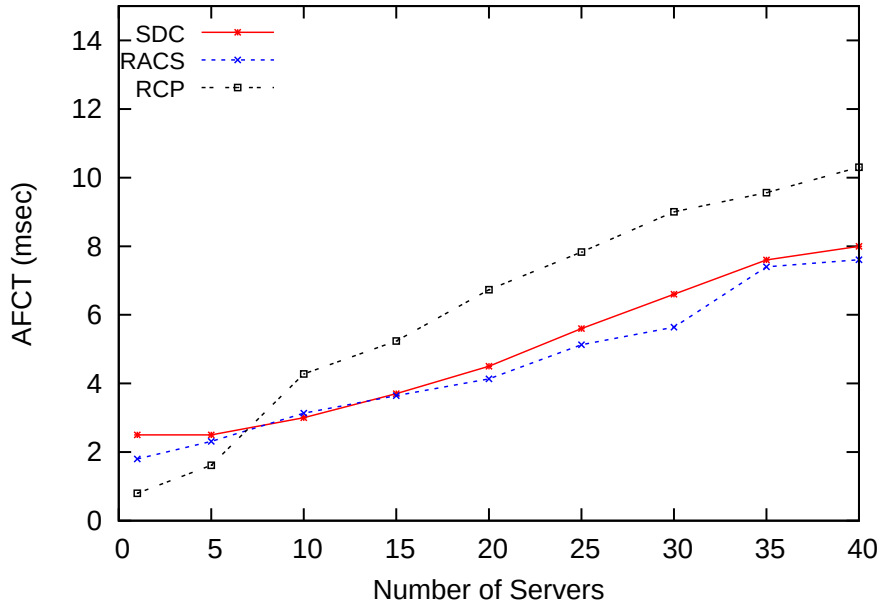

Figure 4.6: Incast: AFCT Comparison

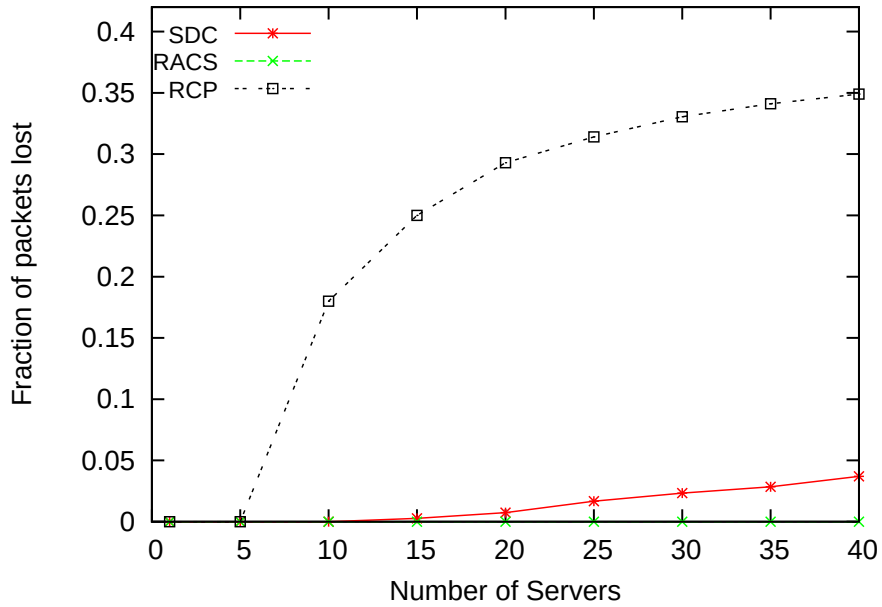

Figure 4.7: Incast: Loss Behavior

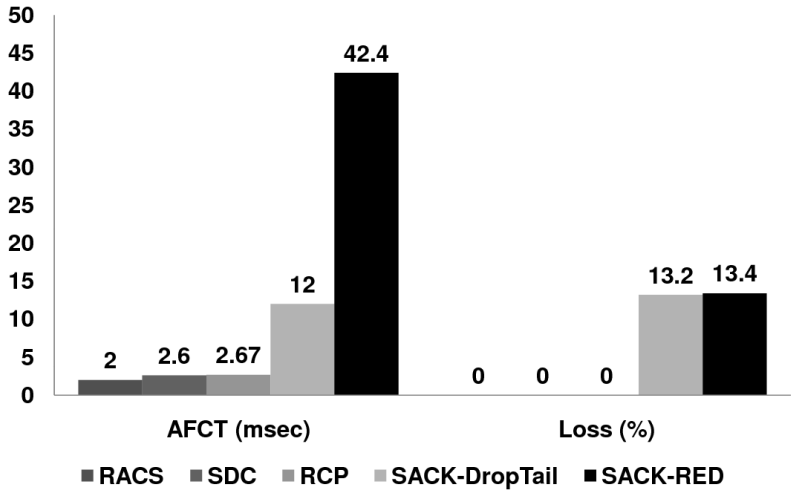

Figure 4.8: Mix Flows: AFCT and Loss Rate

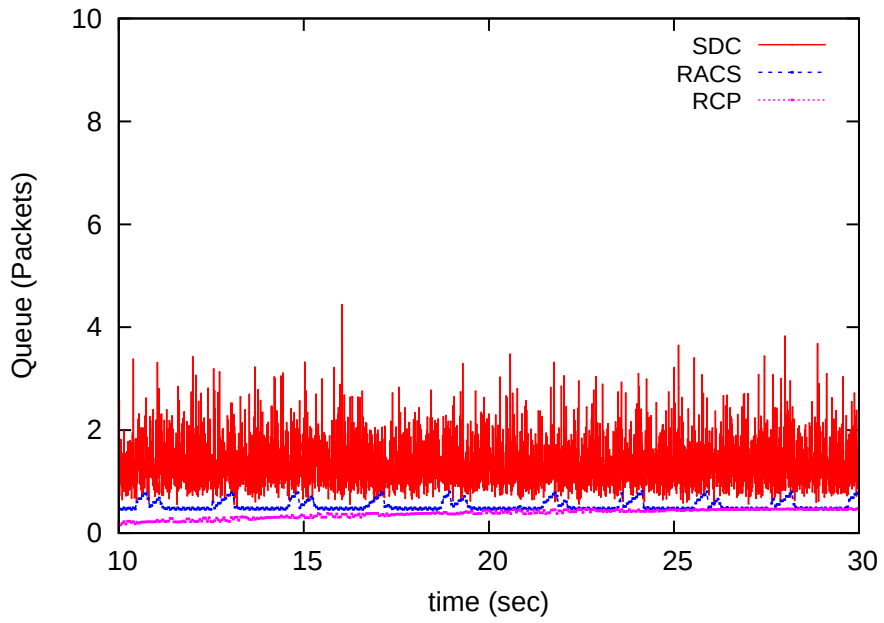

Figure 4.9: Mix Flows: Queue Size (Packets)

	SDC	RACS	RCP
AFCT (msec)	2.6	2	2.67
Utilization(%)	98.11	98.77	98.13
Loss Rate	0	0	0

Table 4.2: RACS vs RCP & SDC

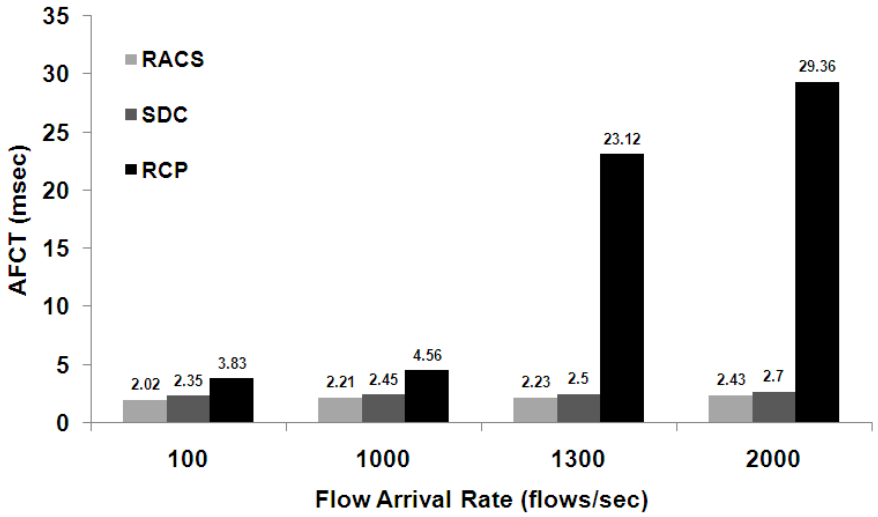

Figure 4.10: Benchmark Traffic: AFCT

4.3.3 Benchmark Traffic

In case of mix of short and long flows we observed that RACS improves AFCT performance significantly. In this section we discuss RACS performance under more general settings common in most data center networks. We generate Pareto distributed traffic with shape parameter 1.2 and average response size of 25KB. To have all mixes of flows, we generate two long flows in the background. Figure 4.10 shows that three protocols show comparable behavior at low loads, but as the arrival rate of flow increases, AFCT of RCP flows increases significantly, largely because of large queue buildups and increased loss rate. On the other hand, performance of RACS largely remains same even at arrival rate of 2000 flows/sec.

It is very important to know the performance improvement regions of RACS over other protocols. For this purpose we compare RCP and RACS,

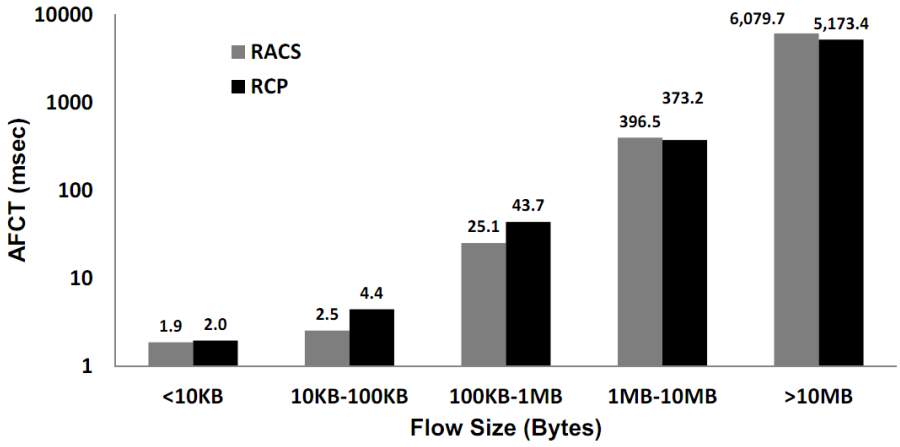

Figure 4.11: Benchmark Traffic: Performance Regions

as RCP is the lower bound on the achieveable gains by other protocols. Figure (4.11), shows that RACS outperforms RCP over a large set of flow size that represent major portion of DC traffic. But for long flows $>$ 1MB we observe throughput degradation in case of RACS. But as our main goal is to improve AFCT of the system, thus we have to compromise on this small price.

To highlight the gains in-terms of how many flows complete before a certain time, we present Figure(4.12) & Figure (4.13). We can see, Figure(4.12), that $\approx$ 98% flows complete within 10 ms in case of RACS and $\approx$ 84% in case of RCP, thus improving application throughput by $\approx$ 20%.

4.4 Summary

In this chapter, we presented design and evaluation of a network assisted congestion control protocol, RACS, designed specifically for the DC networks. Results show that using weighted processor sharing at the routers and insights from scheduling theory can significantly enhance the performance of the network.

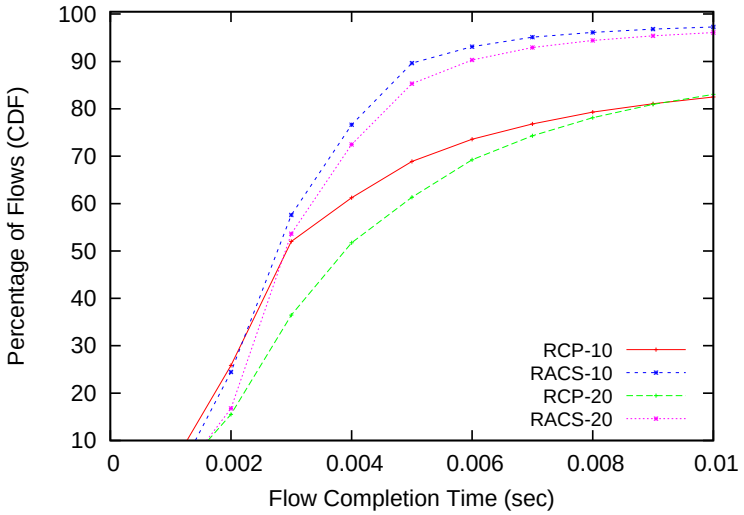

Figure 4.12: Percentage of Flows Completing before Time 't'

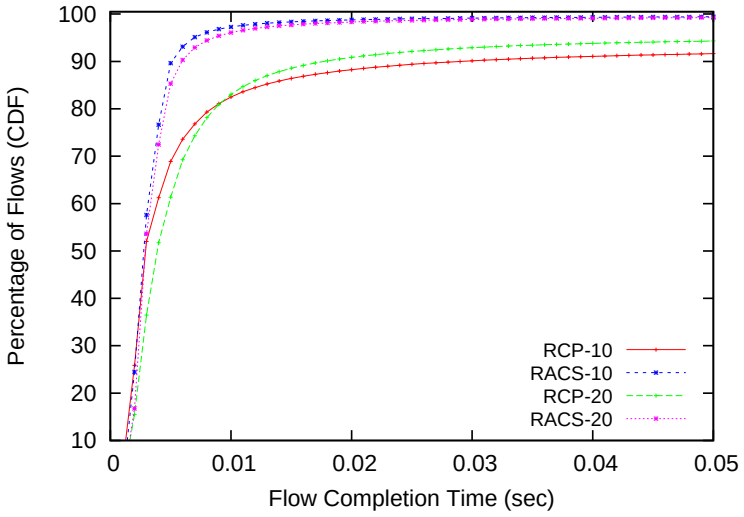

Figure 4.13: Percentage of Flows Completing before Time 't' II

Chapter 5

Conclusion

5.1 Conclusion

The main aim of this thesis was to provide a framework that minimizes the average flow completion time (AFCT) within a data center (DC) network. In this regard we made two contributions, summarized below:

First, we presented the design, analysis and evaluation of a size aware delivery control protocol (SDC). SDC uses flow size information to assign different priorities to the flows. Flows with smaller size are assigned a class with higher α, β parameters, in other words higher priiority. This makes short flows more aggressive than long flows. At routers we use load factor as a congestion signal, which helps us achieve low queue occupation and high link utilization in DC networks. Using analytical modeling, we show that using only three classes can provide significant gains in terms of AFCT over other TCP based protocols [1, 31]. We also observe that benefits of SDC can be improved by using large number of classes with higher α, β parameters, but it also increases the complexity of system as we have to manage more classes and their parameters. Another challange with SDC is that it does not approximate SRPT in a better way, because we use fix α, β parameters within a class which is against SRPT notion. Thus, it is important to explore other approaches that better approximate SRPT.

Second, keeping in view above concern we presented the design and evaluation of a router assisted capacity sharing mechanism (RACS). At sender, RACS assigns a weight to the flow based on its residual size. RACS, employs weighted processor sharing at routers to share capacity between flows. We show that RACS can improve the AFCT of DC network significantly and

it also shows that SDC performance can be improved by using a better approximation of SRPT. We test RACS & SDC under various settings similar to a DC environment. RACS' rate selection mechanism mitigates incast significantly as compared to traditional schemes. Also, being a rate feedback protocol, its convergence time is better than SDC, DCTCP [1] and other TCP variants.

5.2 Future Work

As part of the future work, it would be useful to evaluate the performance of RACS using a real implementation. RACS end-host functionality can be implemented as a loadable kernel module. New congestion control protocols are often implemented as kernel modules to avoid the need to patch the kernel and the subsequent kernel recompilation. The RACS router functionality can be implemented in Linux, NetFPGAs, etc. The control plane can be used to estimate sum of weights and data path can be used to allocate rates based on the weight of a flow.

Bibliography

[1] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pages 63–74. ACM, 2010.

[2] A. Odlyzko. The internet and other networks: Utilization rates and their implications. *Information Economics and Policy*, 12(4):341–365, 2000.

[3] A.M. Odlyzko. Internet tv: Implications for the long distance network. *Internet Television, E. Noam, J. Groebel, and D. Gerbarg, eds., Lawrence Erlbaum Associates*, pages 9–18, 2003.

[4] T. Benson, A. Akella, and D.A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th annual conference on Internet measurement*, pages 267–280. ACM, 2010.

[5] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, Aug 1988.

[6] I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *PFLDNet'05*, February 2005.

[7] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM*, Aug 2002.

[8] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. Procesor sharing flows in the internet. In *IWQoS*, Jun 2005.

[9] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D.G. Andersen, G.R. Ganger, G.A. Gibson, and B. Mueller. Safe and effective fine-grained tcp retransmissions for datacenter communication. *ACM SIGCOMM Computer Communication Review*, 39(4):303–314, 2009.

[10] Y. Chen, R. Griffith, J. Liu, R.H. Katz, and A.D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73–82. ACM, 2009.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72. ACM, 2007.

[13] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[14] T. Hoff. Latency is everywhere and it costs you sales how to crush it, July 2009. http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[16] D. Beaver, S. Kumar, H.C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebooks photo storage. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–8. USENIX Association, 2010.

[17] I.A. Rai, G. Urvoy-Keller, and E.W. Biersack. Analysis of las scheduling for job size distributions with high variance. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 218–228. ACM, 2003.

[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. In *IEEE/ACM Trans. Networking, 1(4):397-413*, Aug 1993.

[20] S. Athuraliya, V. Li, S. Low, and Q. Yin. REM: Active queue management. In *IEEE Network, 15(3):48-53*, May 2001.

[21] L.E. Schrage and L.W. Miller. The queue m/g/1 with the shortest remaining processing time discipline. *Operations Research*, pages 670–684, 1966.

[22] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal. Srpt scheduling for web servers. In *Job Scheduling Strategies for Parallel Processing*, pages 11–20. Springer, 2001.

[23] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[24] N. Bansal and M. Harchol-Balter. *Analysis of SRPT scheduling: Investigating unfairness*, volume 29. ACM, 2001.

[25] R.W. Wolff. *Stochastic modeling and the theory of queues*, volume 14. Prentice hall Englewood Cliffs, NJ, 1989.

[26] M. Nuyens and A. Wierman. The foreground–background queue: a survey. *Performance evaluation*, 65(3):286–307, 2008.

[27] ns-2 Network Simulator. http://www.isi.edu/nsnam/ns/.

[28] S.J. Yang and G. De Veciana. Enhancing both network and user performance for networks supporting best effort traffic. *Networking, IEEE/ACM Transactions on*, 12(2):349–360, 2004.

[29] T. Ziegler, H.T. Tran, and E. Hasenleithner. Improving perceived web performance by size based congestion control. *NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pages 687–698, 2004.

[30] B. Doytchinov, J. Lehoczky, and S. Shreve. Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *Annals of Applied Probability*, pages 332–378, 2001.

[31] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. Technical report, Technical Report MSR-TR-2011-66, Microsoft Research, 2011.

[32] I.A. Qazi, T. Znati, and L.L.H. Andrew. Congestion control using efficient explicit feedback. In *INFOCOM 2009, IEEE*, pages 10–18. IEEE, 2009.

[33] Robert Shorten, Fabian Wirth, and Douglas Leith. A positive systems model of TCP-like congestion control: Asymptotic results. *IEEE/ACM Trans. Networking*, 14:616–629, 2006.

[34] Martin Corless and Robert Shorten. Deterministic and stochastic convergence properties of aimd algorithms with nonlinear back-off functions. *Automatica*, 2011.