# Improving Flow Completion Time and Throughput in Data Center Networks

by

**Sijo Joy**

A Thesis submitted to the Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

**MASTER OF APPLIED SCIENCE**

in Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

**University of Ottawa**

Ottawa, Canada

February 2015

# Abstract

Today, data centers host a wide variety of applications which generate a mix of diverse internal data center traffic. In a data center environment 90% of the traffic flows, though they constitute only 10% of the data carried around, are *short flows* with sizes up to a maximum of 1MB. The rest 10% constitute *long flows* with sizes in the range of 1MB to 1GB. Throughput matters for the *long flows* whereas *short flows* are latency sensitive. This thesis studies various data center transport mechanisms aimed at either improving flow completion time for *short flows* or throughput for *long flows*. Thesis puts forth two data center transport mechanisms: (1) for improving flow completion time for *short flows* (2) for improving throughput for *long flows*. The first data center transport mechanism proposed in this thesis, FA-DCTCP (Flow Aware DCTCP), is based on Data Center Transmission Control Protocol (DCTCP). DCTCP is a Transmission Control Protocol (TCP) variant for data centers pioneered by Microsoft, which is being deployed widely in data centers today. DCTCP congestion control algorithm treats *short flows* and *long flows* equally. This thesis demonstrate that, treating them differently by reducing the congestion window for *short flows* at a lower rate compared to *long flows,* at the onset of congestion, 99$^{th}$ percentile of flow completion time for *short flows* could be improved by up to 32.5%, thereby reducing their tail latency by up to 32.5%. As per data center traffic measurement studies, data center internal traffic often exhibit predefined patterns with respect to the traffic flow mix. The second data center transport mechanism proposed in this thesis shows that, insights into the internal data center traffic composition could be leveraged to achieve better throughput for *long flows*. The mechanism for the same is implemented by adopting the Software Defined Networking paradigm, which offers the ability to dynamically adapt network configuration parameters based on network observations. The proposed solution achieves up to 22% improvement in *long flow* throughput, by dynamically adjusting network element's QoS configurations, based on the observed traffic pattern.

*Dedicated to the infinite power that Creator has put to work in all of us.*

# Acknowledgement

The kindness and insights from various quarters have given me the impetus for the path traversed towards my thesis. Recognition and my deepest gratitude go to my Professor and Research Supervisor Dr. Amiya Nayak, for his constant support and critiques on my work. The works of Rob Sherwood and Monia Ghobadi have inspired me, and the interactions with them, though few helped me add dimensions to my research. I thank Robert McMahon for the support and help with Iperf tool customizations. I am truly and deeply indebted to my friends Breeson Francis, Rakesh Kappoor, Iype P Joseph, David Perry, Dr. Paul Heintzman and Visakh Kanavallil, the numerous motivating and inspiring discussions that I had with them and their camaraderie have helped me much in this journey. I cannot thank enough my parents who are always there for me and who were the most thrilled at my decision to pursue graduate studies. I am happy that I have my wife, Surya, and dear son, Roy, to applaud me always. Without their love and support, I would be nowhere. As in the words of Paulo Coelho, Roy shows me every day to be happy for no reason, to be curious and to strive tirelessly.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| APIs | Application Programming Interfaces |
| AQM | Active Queue Management |
| CPU | Central Processing Unit |
| D2TCP | Deadline-aware Data Center TCP |
| D3 | Deadline Driven Delivery |
| DCTCP | Data Center Transmission Control Protocol |
| DIBS | Detour Induce Buffer Sharing |
| DIPs | Direct IP addresses |
| ECMP | Equal Cost Multi Path protocol |
| ECN | Explicit Congestion Notification |
| EDF | Earliest Deadline First |
| FA-DCTCP | Flow Aware Data Center TCP |
| FCT | Flow Completion Time |
| FIFO | First In First Out |
| GB | Giga Byte |
| GHz | Giga Hertz |
| HULL | High-bandwidth Ultra-Low Latency |
| IP | Internet Protocol |
| KB | Kilo Byte |
| L2DCT | Low Latency Data Center Transport |
| LAS | Least Attained Service |
| LOC | Lines of Code |
| LTS | Long Term Support |
| MB | Mega Byte |
| Mbps | Mega bits per second |
| MPTCP | Multi Path Transmission Control Protocol |
| NIC | Network Interface Card |
| OLDI | Online Line Data Intensive |
| OS | Operating System |
| OSPF | Open Shortest Path First |
| OVSDB | Open Virtual Switch Data Base |
| PC | Personal Computer |
| PDQ | Pre-emptive Distributed Quick flow scheduling |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| RED | Random Early Detection |
| RFC | Request For Change |
| RTO | Retransmission Timeout |
| RTT | Round Trip Time |
| SDN | Software Defined Networking |

| | |
|---|---|
| SJF | Shortest Job First |
| SLA | Service Level Agreement |
| TCP | Transmission Control Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TE | Traffic Engineering |
| ToR | Top of the Rack |
| Tx | Transmit |
| VIPs | Virtual IPs |
| VL2 | Valiant Load Balancing |
| VM | Virtual Machine |
| WAN | Wide Area Network |
| XMP | Explicit Multipath Congestion Control Protocol |

# Chapter 1 - Introduction and Motivation

This chapter introduces a reader to the objectives of the thesis and its benefits. It explains motivation and lists the contributions of the thesis research. The chapter wraps up by explaining the organization of this thesis document.

## 1.1. Introduction

Today, cloud data centers house computing platforms that host large web scale modern Internet applications such as search, social networking and advertisements selection. These modern day data centers house thousands of servers interconnected by high bandwidth network - termed as the data center network. Data center network is the central nervous system, that enables the distributed applications hosted in the servers of a data center to communicate and interoperate [1].

Data center traffic is often characterized and measured in terms of flows. A flow corresponds to a sequence of packets, from an application residing on a source to an application on the destination host, and is uniquely identified by the canonical five-tuple (source IP, destination IP, source port, destination port and protocol). Data center traffic measurement studies [2]–[4] have shown that data center traffic is bimodal, comprising of short flows and long flows. Short flows are queries, responses or control messages that get exchanged between servers. Short flows, often termed as mice flows (or foreground flows), are latency sensitive and can be of maximum size 1MB. Short flows constitute 90% of the traffic flows inside the data center, but only 10% of the data being exchanged are in short flows [2]–[4]. Long flows, otherwise called elephant flows, correspond to applications that perform large transfers typically backup, virtual machine migration and data-structure synchronization inside a data center. Long flows can be of sizes ranging from 1MB to about 1 GB. Though long flows constitute only 10% in terms of the traffic flows in the data center they carry the majority (90%) of the data that is being exchanged internally [2]–[4]. Short flows are latency sensitive, since they are often associated to soft real time applications which are bound by tight deadlines stipulated by a service level agreement (SLA). Throughput matters for long flows which are often associated to bulk data transfer.

Data centers hosting online, interactive and data intensive (OLDI) applications exert diverse demands on the data center networking. A significant requirement among them is, ensuring soft-real time latency guarantees expressed in terms of service level agreements (SLA) are met, which is crucial in ensuring user satisfaction and increased return traffic [5], [6]. Google reported 25% reduction in search query traffic, as a result of an increase of 0.5 seconds in search page generation [5], [7]. Amazon experienced 1% reduction in sales, for every 100 milliseconds of latency [6]. Bing observed a dip of 1.8% in search queries per user, due to 2 seconds slowdown in the search page rendering [7]. Shopzilla reported 25% increase in the page views up on 5 seconds speed up in page generation [7]. At the same time, it is imperative to use commodity networking gear, especially commodity switches, to build data center networks, because of the economies of scale [3], [8], [9]. Commodity switches come with shallow buffers which are shared by multiple ports. For example, it is common place to have 30-40 ports sharing 4MB of shared buffer in the Top of the Rack (ToR) switch in a typical data center network topology. Shared and shallow buffered nature of commodity switches make them more vulnerable to packet loss, due to queue overflow resulting from congestion. Another aspect which adds on to the demands on the data center networking is, partition/aggregate structure employed by the online data intensive applications. Partition/aggregate is an application design pattern, where a request is split by the front end server or aggregator and assigned to worker nodes. Aggregator collects the replies from the worker nodes and forms the reply for the user. In such an environment, the time taken for a worker node to send data across to the aggregator significantly impacts the latency perceived by the user. In this scenario, the link from the aggregator to the switch becomes the bottleneck link that would experience network congestion depending on the workload. Generally the congestion happens in two instances: (a) when all worker nodes concurrently send a response that results in a burst of packets which will overwhelm the shared packet buffer at bottleneck link [10], [11] (b) when long flows (background flows), that transfer control information and refresh data for the nodes in OLDI applications, build up large queues in the bottleneck queue buffer. In summary, shallow buffering combined with the partition/aggregate structure of the applications in cloud data center causes: (a) incast impairment: concurrent fan-in burst from worker nodes exceeding the buffer capacity causing congestion collapse at bottleneck link, (b) queue build up: long flows building up queue at the switch interface causing the short flows to

experience longer delays even when there is no loss of packet, and (c) buffer pressure: long flows occupying a major portion of the switch buffer and consequently buffer would not have room to accommodate packet bursts which could result in packet loss.

## 1.2. Motivation

To meet the needs of short flows, with commodity switches in a data center that employs partition/aggregate structure, requires a transport protocol that ensures low queueing delay at switches and thereby low latency and burst tolerance. At the same time, the protocol should guarantee that the high throughput requirements of long flows are met. This exemplifies the fact that, from networking perspective it is important to ensure that the nature of traffic flows inside data center is determined and treated appropriately, ensuring that the requirements of specific traffic type are met.

High performance data center transport mechanism is often characterized as, one which tries to achieve at least one of the following while not compromising on others - minimizing flow completion time, maximizing throughput or reducing deadline miss rate [11], [12]. This thesis focuses on minimizing flow completion time for short flows and maximizing throughput for long flows.

DCTCP [11] is the first transport protocol designed keeping the data center environment in mind. DCTCP employs a mechanism based on Explicit Congestion Notification (ECN) [13], to estimate the extent of congestion. DCTCP then uses this estimate, to scale the congestion window at the onset of congestion, thereby reacting to the extent of congestion, rather than blindly reacting to the presence of congestion, by halving the congestion window as in traditional TCP. This way DCTCP helps to keep the queue occupancy low and at the same time ensures throughput is not affected, by reacting to the extent of congestion, not to the presence of it. Thesis work detailed in chapter 3 shows that, it is possible to go one step further by incorporating flow awareness in to DCTCP. Flow awareness could be used to adapt the congestion algorithm in DCTCP for meeting low latency requirement of short flows in a much better fashion.

Data center traffic measurement studies [2]–[4], shows that internal data center traffic exhibits pronounced Time-of-Day and Time-of-Week patterns. Thesis work detailed in

chapter 4 illustrates that, it is possible to leverage this insight to achieve better throughput for long flows, by employing a dynamic adaptation framework based on Software Defined Networking paradigm.

## 1.3. Objectives and Contributions

Following are the major objectives of the thesis:

The main goal and the theme of the research is, to come up with proposals for data center transport schemes that help to achieve better latency for short flows and better throughput for long flows. Based on this research theme, following are the major objectives of the thesis:

1. Analyze the feasibility of imparting differential treatment to long flows and short flows at the transport layer in data center networks.
2. Analyze the literature and identify mechanisms for improving flow completion time for latency sensitive short flows via differential treatment of data center traffic flows.
3. Explore the literature to figure out the feasibility of a mechanism to improve throughput for bandwidth sensitive long flows.
4. Analyze and compare the achieved results with existing mechanisms.

Following are the major contributions of the thesis:

1. Analyzed the feasibility of imparting differential treatment to long flows and short flows at the transport layer in data center networks and identified two mechanisms for the same from literature.
2. Analyzed the literature and based on the insights garnered proposed a data center transport mechanism - Flow Aware Data Center TCP (FA-DCTCP) - for improving flow completion time for latency sensitive short flows, via differential treatment of data center traffic flows.
3. Put forth a proposal to improve throughput for bandwidth sensitive long flows, via dynamic adaptation of ECN threshold using the SDN paradigm.
4. Demonstrated improvement of up to 32.5% over DCTCP, for the 99$^{th}$ percentile of Flow Completion Time, for short flows via the proposed FA-DCTCP mechanism. Achieved up to 12% improvement over DCTCP and 22% improvement over ECN* for

long flow throughput via the proposed SDN based dynamic ECN adaptation framework.

The following paper [14], based on the contribution FA-DCTCP from the thesis, is accepted for publication at the IFIP/IEEE IM 2015 conference.

S. Joy and A. Nayak, "Improving Flow Completion Time for Short Flows in Data Center Networks," in IFIP/IEEE International Symposium on Integrated Network Management (IM 2015), 2015

The following paper, based on the contribution SDN based dynamic ECN adaptation scheme, is in submission at the IEEE ICC 2015 conference

S. Joy, A. Nayak, "SDN based ECN Adaptation Scheme for Improving Data Center Long Flow Throughput", IEEE ICC Workshop on Smart Communication Protocols and Algorithms (ICC 2015), June 2015, Submitted

## 1.4. Organization of Thesis

This thesis is organized in five chapters. Chapter 1 provides an outline of the research to the reader. It also presents the objectives of the work, its benefits and motivation. Chapter 2 provides a background about the technologies used in the work: data center networking, data center transport protocols and software defined networking. Chapter 2 also provides a review of the data center transport protocols. Chapter 3 presents FA-DCTCP proposal, an improved version of DCTCP for achieving better latencies for short flows while ensuring the throughput of long flows are not affected. Chapter 4 describes SDN based dynamic ECN adaptation framework, aimed at improving throughput for long flows. Chapter 5 concludes the thesis, by giving a summary of the research and future research directions based on the thesis.

# Chapter 2 - Background and Related Work

This chapter presents the background on technologies that are used in the thesis research. It introduces the reader to Data Center Networking, Software Defined Networking and presents literature survey on Data Center Transport protocols.

## 2.1. Data Center Networking

Data centers are the building blocks of modern day computing, providing a variety of information technology services: including web-searches, e-commerce and social networking. A consistent theme in data centres today is to take advantage of commodity hardware for compute, storage and network. Economics warrant that commodities of scale can be more favorable to businesses and this has led to the emergence of "mega data centres", hosting cloud applications running on servers to the tune of thousands [15].

Cloud applications involve substantial data processing and large scale distributed computations. Data centre networks, comprising of tens of thousands of machines, interconnected with commodity switches are built to meet these requirements [16]. Data center networks facilitate many facets of online services that are Internet-facing "sensitive" applications like web services, instant messaging, online media sharing, financial applications and gaming, to computationally intensive applications like indexing web content, data analysis, scientific computing and various others. All these applications rely on data center's network infrastructure for optimal performance. In the case of a congested data center network, internal traffic is invariably under pressures of packet losses and poor throughput. This results in instances of connection snaps, heavy lags, and suspension of essential tasks such as web transactions, that can ultimately effect disgruntled end users and may affect the revenues [17], [18].

A data centre network is to a data center, is what a central nervous system is to a human body [19]. Internode communication network and its bandwidth is a key bottleneck in a modern day data centers housing large scale clusters. Most of the applications that are resident in data centers today, are required to communicate and exchange data and information with other nodes to progress with their local computation, and therefore exert

significant demands on the network that ties the data center components together [8]. For example, in order to respond to a search query, web search engine are often required to partition the work and query all the nodes in the cluster hosting the inverted search index, which requires parallel communication between the participating nodes [20]. Service oriented architecture [21], employed by Internet services requires - coordination and communication with hundreds of remote nodes providing sub-services - in order to retrieve a webpage. Parallel scientific application, another predominant application hosted in data centers today, require significant inter node communication [22], [23]. In addition, modern day distributed computing frameworks - Hadoop [24], Dryad [25], and MapReduce [26] and web services - ecommerce, social networking and search, necessitated the construction of massive scale-out computing clusters, composed of commodity servers and commodity networking elements, such as commodity of the shelf switches and routers [27].

The two main approaches for building data center networks are:

(a) **Leveraging specialized communication protocols and hardware**, for example Myrinet [28], InfiniBand [29]. Though this approach can scale thousands of nodes with high bandwidth, they are more expensive, since they do not leverage commodity parts. Since this approach is based on specialized communication protocols, it is not natively compatible with TCP/IP applications.

(b) **Making use of the commodity networking components**, such as commodity of the shelf Ethernet switches and routers, to achieve economies of scale. This approach makes it possible to retain a familiar management infrastructure and do not require custom modification to applications, operating systems, and hardware. Inability to scale cluster bandwidth in line with cluster size is a downside of this approach. Data center network architecture proposals such as Monsoon [30], VL2 [3], Fat-Tree [8], BCube [9], PortLand [15], Hedera [27], etc. try to address this aspect, by using commodity network elements and adopting multi-path routing. Majority of the data center network deployments today follows the second approach, due to cost and compatibility reasons. Focus of the thesis is on data center network built out of commodity components following the second approach.

One of the underlying observations used in this thesis work is that, current day data center networks are often based on multi rooted tree topology [31] with bandwidth

aggregation at different levels of the network. Canonical data center architecture, from Cisco Data Center Infrastructure 2.5 Design Guide [31] (a recommended source for data center network design), is depicted in Figure 1. The topology a variant of a fat-tree (or folded-Clos topology) as captured in Figure 1, has an aggregate bandwidth that grows in proportion to the number of host ports in the system. The main motivation, for adopting such a topology, is the topology's inherent ability to meet the most important requirement for a scalable network, where increasing the number of ports in the network facilitates linear increase in the delivered bisection bandwidth.

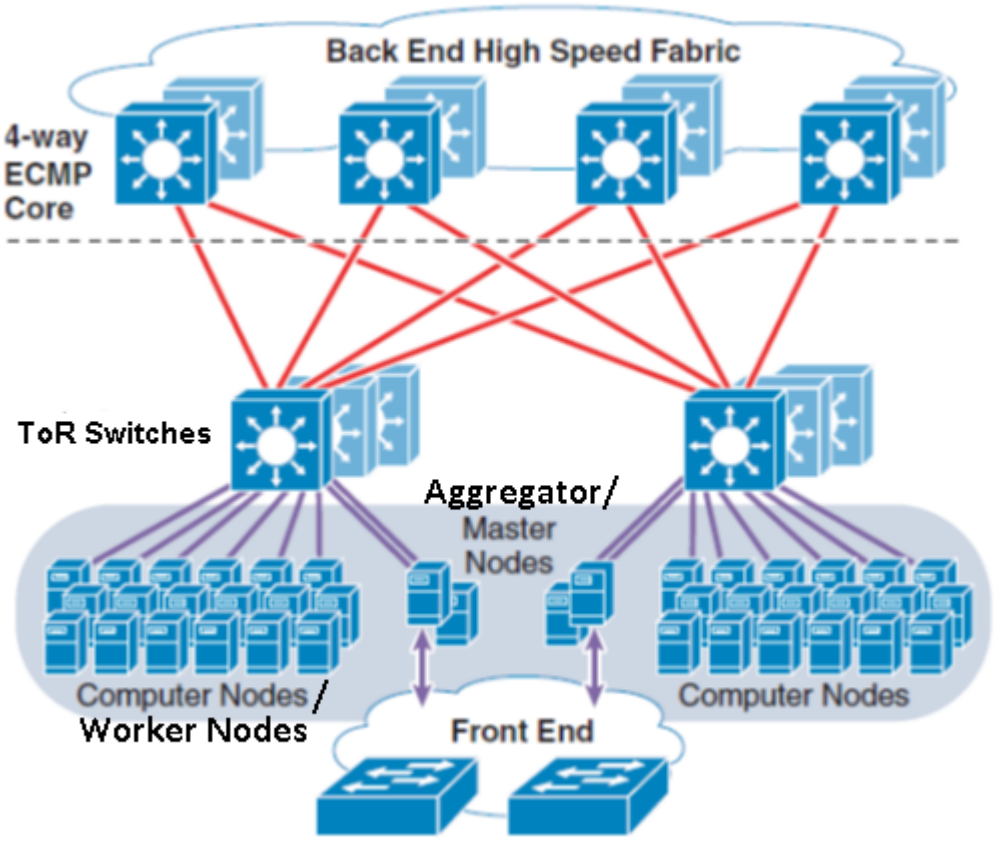


Figure 1: Canonical Data Center network architecture adapted from Cisco Data Center Infrastructure 2.5 Design Guide [31]

Multiple applications run inside a typical data center. Each of these applications will be hosted on its own set of (possibly virtual) server machines. A publicly visible and routable IP address is associated with each of these applications, to which clients in the Internet direct

their requests. Load balancer which owns the publically visible IPs (termed as Virtual IPs - VIPs) inside the data center, spreads the requests among a pool of front-end servers that process the requests. The IP addresses that correspond to the servers on to which the request are spread are known as direct IP addresses (DIPs). Requests from Internet are directed towards the destination VIP, by layer 3 routing through border and access routers to a layer 2 domain which hosts the load balancers and the end servers. The VIP is configured onto the two load balancers connected to the top of the rack switches, and redundancy mechanisms are used to ensure that if one load balancer fails, the other takes over the traffic. For each VIP, the load balancers are configured with an associated list of DIPs that corresponds to the private internal addresses of physical servers in the racks below the load balancers. Load balancer distributes requests across the DIPs in the pool that corresponds to a VIP depending on the load on the servers [30]. Another application pattern with the same essence at its heart is the partition/aggregate pattern employed by web search, social networking applications. A request made by a higher layer application will be dispatched by an aggregator to the lower layers or workers after breaking down the request. The final coalescence of response from individual worker layers is used in answering the request.

The usage of commodity network elements in data center network has significant implications on the data center transport design. Those implications are due the competing demands exerted by the prevalent application patterns on the data center network. Following sections discusses them and the subsequent section on data center transport will discuss strategies employed in data center transport design to address these demands.

## 2.2. Data Center Traffic

Cisco global cloud index [32], offers interesting insights in to the growth of global data center and cloud-based IP traffic. It underlines the evolution of cloud adoption in the telecommunications industry from an emergent technology to an established networking solution with ubiquitous acceptance and high percolation globally. Cloud services offers absolute access, with almost no restriction of location, to contents and services for the end users: the consumers across multiple devices. This ease of access with flexibility has made businesses take off major portions of their mission-critical workloads on to a cloud from the

conventional realms of test environments. There can be no denying the laterality of cloud computing on data center traffic.

The advent of data centre paved way for Internet traffic that either originates or terminates in a data center. It has been gaining further momentum through the years and the dominance of data center traffic appears to be unabated for the foreseeable future. Parallelly, there has been a fundamental transformation of the nature of data center traffic impacted by cloud applications, services, and infrastructure. The three broad classifications of consumer and business traffic flowing through data centers are:

a) **Traffic that remains within the data center:** A major chunk of the data center related traffic remains within the data center, because of factors such as the functional separation of storage, application servers, and databases, which generate internal traffic that traverses the data center because of replication, backup, and read and write traffic between the three components. Writing data to a storage array, or moving data from a development environment to a production environment within a data center are examples of internal data center traffic. Also parallel processing schemes like partition/aggregate patterns employed in data centers divide tasks and send them to multiple servers, contributing to internal data center traffic. Cisco global cloud index [32] indicates that 76.7% of the data center traffic belongs to this category.
b) **Traffic that flows from data center to data center:** 6.6% of data center traffic falls in to this category. Examples of inter data center traffic are copying content to multiple data centers as part of a content distribution network, or moving data between clouds [32].

c) **Traffic that flows from the data center to end users through the Internet or IP WAN:** 16.7% of data center traffic belongs to this category. Streaming video content to end users PCs or mobile devices is an example of traffic flow from data center to end users on the Internet.

Focus of the thesis is on the data center transport protocols that deals with the first category of data center traffic, which is the traffic that remains within the data center. It is to be noted that data center often employ schemes to effectively shield and separate the three

classes of traffic from each other, by deploying load balancers, that separate traffic that leaves the data center from the traffic that stays within the data center [11].

Data center traffic measurement studies [2]–[4], point out that the data center environment differ significantly from wide area networks. Round Trip Time (RTT) within the data center are of the order of few milliseconds. Application mix within the data center often exert competing demands on the data center network, a pronounced requirement is the ability to offer the trade-off required to accommodate traffic mix that requires very low latencies and high throughput at the same time.

### 2.2.1. Communications in Data Centers

Current web applications such as search, social networking, and document collaboration hosted in data centers rely on services sourced from multiple clusters. Multiple applications are hosted by these clusters at the same time, to increase overall system utilization. User-facing applications have soft real-time latency guarantees or SLAs, often to the tune of tens of milliseconds that the application must meet. In order to tackle this, many applications employ hierarchical multi-layer partition/aggregate pattern work flow, depicted in Figure 2, where a user request is subdivided and dispatched to worker threads within the cluster. The worker threads generate replies and these replies are aggregated and returned to the user. In such a setting, latency of one layer can affect the other layers. For example, in the case of web search, a query will be directed by the aggregators to workers which are responsible for a different part of the database index. Any lag in this partition/aggregate can delay the response time associated to queries, thus making the 99.9th percentile of latency (tail latency) a vital aspect of data centers [11].

Recent measurements show that 99% of current data center traffic is TCP traffic [11]. Even though TCP is a mature technology that has evolved over the years with optimizations and variants that meet the communication needs of most applications, the workload, scale and environment of data centers is quite different than the WAN environment related assumptions that TCP was originally based on. A case in point is the default TCP retransmission timeout (RTO) timer value in contemporary operating systems. RTO timer value is set to 200ms by default in Linux operating system, which is reasonable for WAN

environment, but it is 2-3 times order of magnitude greater than the average data center RTT [10]. This causes data center network specific impairments, like TCP throughput collapse termed as incast collapse [34] in data center literature, which results in gross under-utilization of link capacity in data center employing partition/aggregate patterns, and in-turn demands customized data center transport protocols tailored for the data center environment.

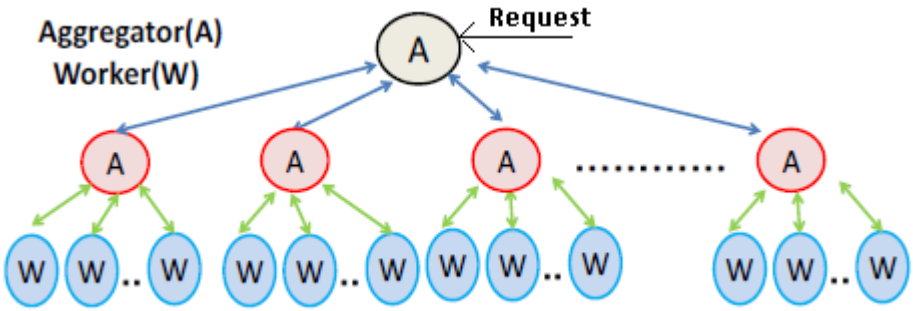


Figure 2: Partition/Aggregate design pattern adapted from [33]

Xu et al. [35] points out the following, about application characteristics, and their expectation from the data center network, in their research paper on reducing tail latency in cloud data centers. Modern day web applications such as Facebook, Bing, Amazon etc which are dynamic and interactive relay on large scale data centers with number of nodes crunching large data sets. Hundreds of worker nodes (servers) are contacted for generating a single page view in such sites [21], and a delay in response from any one of them could lead to significant end-to-end delay [11] which could lead to poor end user perception about the site [36]. In such setting, latency is considered the most important aspect to attain [37], [38] and the long tail of latency is of much significance, since the 99.9th percentile of network RTTs are orders of magnitude worse than the median [11], [39], [40]. In such a system, one out of thousand customers will experience an unacceptable lag in page load performance.

## 2.2.2. Workload Characterization

Previous section covered the communication patterns in data centers. This section summarises the insights from previous research studies, highlighting the aggregate data center network transmission behaviour.

Alizade et al. [11] present measurements of a 6000 server Microsoft production cluster, to suggest that mainly two types of traffic co-exist in cloud data centers: throughput sensitive long flows (1 MB to 1GB) and latency sensitive short flows (1KB to 1MB). It is to be noted that these two types of flows exert conflicting requirements on link buffer occupancy[11], [16]. Data center traffic studies [2]–[4] illustrate that that long flows correspond to bulk data transfer, such as data-structure synchronization, data storage synchronization, virtual machine backup, virtual machine migration, replication, data mining etc. Short flows often correspond to distributed applications like MapReduce, web search queries, control traffic [2]–[4]. Long flows strive to achieve high throughput, leading to higher buffer occupancy at the bottleneck link buffers. At the same time short flows strive to attain better latency, by virtue of being constrained by latency sensitive applications [11], [40]. [11] points out that in-order to attain low latency, link buffer occupancy should be kept low, which leads to the trade-off between throughput and latency, which the data center network and transport protocols should help in achieving [16].

Benson et al. [2] measure the network traffic in 10 data centers, and suggest that most flows in the data centers are short (up to a maximum of 1MB), and majority of them last for a few hundreds of milliseconds. Another observation is that, irrespective of the differences in the size and usage of the data centers, traffic traversing the racks in a data center environment are bursty (ON/OFF) and have properties that fit heavy-tailed distributions. It goes on to observe that, in the cloud data centers, a major portion of the traffic originated by servers (80%) stay within the rack, in most data centers irrespective of the type, link utilizations are rather low in the ToR and aggregate. The exact number of highly utilized core links varies over time, which often stays within 25% of the core links in any data center. It also notes that, the losses within the data centers are not localized to highly utilized links; instead they often manifest at the low average utilization links, indicating momentary bursts as the primary reason. The magnitude of losses is greater at the aggregation layer than at the edge or the core layers, because of the bandwidth oversubscribed nature of data center network topologies based on multi-rooted trees. Benson et al. [2] further expounds that, the link utilizations are subject to Time-of-Day and Time-of-Week effects across all data centers.

## 2.3. Data Center Transport Mechanisms

This section provides a high level overview of data center transport protocols. This section highlight the need for mechanisms that leverages flow awareness and data center traffic patterns, while retaining the simplicity and ease of deployment offered by a solution requiring minimal end-host changes and no changes to the network fabric.

Number of data center transport protocols has been proposed in recent years, to cater to the specific demands data center environment pose. These data center transport protocols aim to improve the performance of cloud applications, by explicitly accounting for the network and traffic characteristics prevalent in data centers. Data center transport mechanism are devised with at least one of the following objectives: (a) minimize flow completion time (b) maximize throughput, or (c) reduce deadline miss rate [11], [12]. To achieve their objective, data center transport mechanism often employ techniques that target one or a combination of different artifacts in the data center network, including queue occupancy reduction, traffic burst smoothening out (there by avoiding buffer pressure), avoidance of congestion collapse due to synchronized senders (incast collapse) etc. Existing data center transports belong to one of the following four transport strategies (classified by the underlying technique employed by the protocol to achieve their objective): (a) Self-Adjusting Endpoints, (b) In-network Prioritization, (c) Arbitration, or (d) Leveraging multipath nature of data center network. Below section discusses these strategies and build the premise for the thesis research contributions.

### 2.3.1. Self-Adjusting Endpoints

The strategy employed by self-adjusting end points schemes are modeled similar to traditional TCP. [11] points out that majority of flows in a data center are TCP flows, so TCP based schemes are the prominently deployed transport schemes in data centers [41]–[43]. The crux of this transport strategy is that the end-points makes decisions about the amount of data to be send based on network congestion indicators. These schemes employ either, implicit (e.g. packet loss) or explicit (e.g. ECN) congestion signalling mechanisms, to determine the network congestion state. At the onset of congestion, the congestion window at the sender is adapted (modulated) based on the core objective of the schemes. For example,

if the core objective is (a) fairness: congestion windows are scaled down by the same extent for all flows [11], [44] (b) flow prioritization: congestion windows are scaled down by remaining amount of data to be transferred for each flows [45] or by the deadlines associated to the flows [46]. Protocols belonging to this category do not require any changes in the network infrastructure, since it confine the implementation aspects to the endpoints and hence are deployment friendly.

#### 2.3.1.1. DCTCP

Data Center TCP: DCTCP [11] is a TCP variant for data center environments that aim to reduce the queue occupancy making use of an active queue management mechanism - ECN [13]. ECN is now common and is supported in all commodity switches that are deployed in data centers. DCTCP requires implementation changes only at the end hosts and a simple configuration change at the switches. It also detects the onset of congestion early by monitoring of the egress queues of the switches and by ECN marking the packets when the queue occupancy reaches a prior configured threshold. DCTCP marking is done based on the instantaneous queue occupancy, depicted in Figure 3, which helps to identify and react to congestion at the early stages of congestion buildup.

DCTCP reacts to the extent of congestion rather than to the presence of congestion by deriving multi bit feedback from the ECN marked packets at the source of the traffic. DCTCP scales the congestion window in such a fashion that, it is reduced in proportion to the congestion experienced in the network, rather than halving the window as done in traditional TCP. This way DCTCP ensures high throughput while at the same time low queue occupancy is also achieved. This in-turn facilitates lower latency for short flows.

A main point to note here is that, DCTCP treat short and large flows in the same manner; in the sense that the congestion window pertaining to short and long flow windows are reduced to the same extent, and is driven by the ECN marked packets received from the source for that particular flow. DCTCP is the pioneering work that triggered the development of further specialized transport protocols for data center networking. It stands out by the simplicity offered in implementation and the ease of deployment. DCTCP is available in Microsoft windows server 2012 and is deployed in data centers. The mechanism proposed in this paper is based on DCTCP and incorporates flow awareness into it.

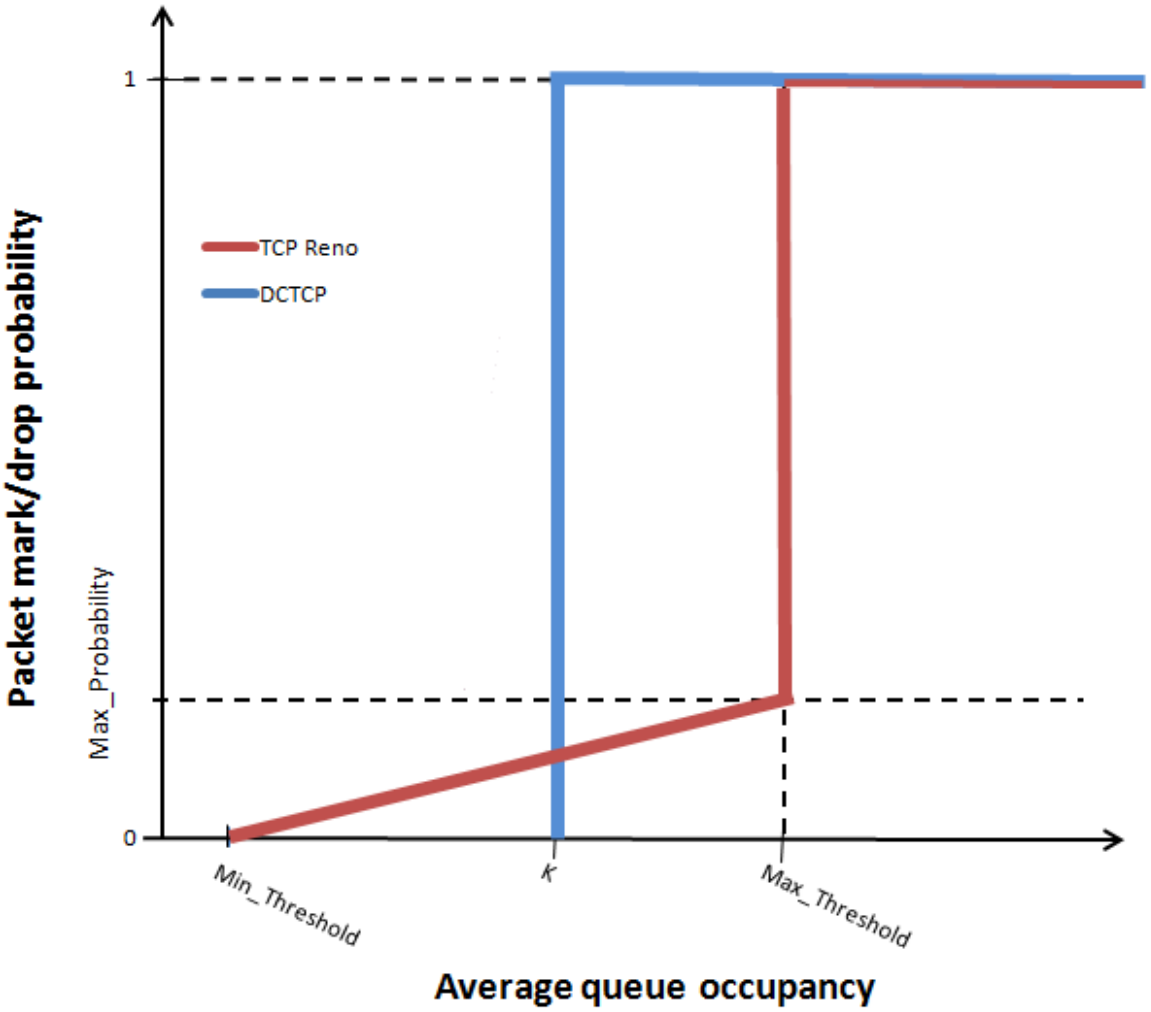


Figure 3: DCTCP AQM scheme is a variant of Random Early Detection (RED). Low and high marking threshold are set to the same value, ensuring packet marking and congestion notification based on instantaneous queue length

### 2.3.1.2. ECN*

TCP+ECN scheme based on instantaneous queue length at the switches: Wu et al. [44] proposed a hybrid AQM scheme, ECN*, for data centers which is based on the idea of using a simple ECN marking scheme (based on instantaneous queue occupancy at the bottleneck switches) to adapt TCP congestion window. ECN* proposes guidelines for choosing the lower bound and upper bound for marking threshold, considering the trade-off between throughput and latency for data center flows. This work shows that ECN marking threshold parameter setting can be tuned to achieve the desired trade-off, but the implementer has to tune the parameter to meet the specific requirement (for example low latency), while the

other (in this case throughput) would experience degradation. Chapter 4 is motivated by this work, and extends it to dynamically adapt the marking threshold at the switches based on the traffic patterns in the network.

#### 2.3.1.3. D2TCP

Deadline-aware Data Center TCP: D2TCP [46] is a data center transport protocol that adds deadline awareness to TCP's reactive and distributed approach, in order to better meet OLDI deadlines, achieve high bandwidth for long flows while ensuring ease of deployment and TCP compatibility. D2TCP congestion avoidance algorithm makes use of gamma-correction function, with ECN feedback and deadline information as inputs, to manipulate the congestion window size. By adding deadline awareness in to congestion avoidance algorithm, D2TCP prioritizes near-deadline flows in the presence of congestion and ensures that far-deadline flows back off aggressively; whereas near-deadline flows back off only a little or not at all. D2TCP does not require changes to in-network elements and could be deployed by upgrading the TCP and RPC stacks on end-hosts. Evaluation results presented in [46] show that D2TCP reduces fraction of missed deadlines compared to DCTCP by 75%.

#### 2.3.1.4. L2DCT

Low Latency Data Center Transport: L2DCT [45], proposes a data center transport mechanism, which updates the congestion window emulating the Least Attained Service (LAS) scheduling discipline. Mechanism prioritizes flows by incorporating a mechanism that does not require the deadline information a priori. The congestion window adaptation decisions are based on the amount of data already sent by a particular flow. Congestion window adaptation rate is determined by weights associated with a flow, which in turn corresponds to the amount of data the flow had already transmitted. This mechanism implies that long flows back off aggressively than short flows in the presence of congestion. This scheme also ensures that, in the absence of congestion long flows increase their window at a lower rate in comparison to short flows. L2DCT reduces the mean flow completion time by 50% compared to DCTCP and 95% compared to TCP. However a main weakness of L2DCT is that, even for low data center traffic loads of about 10%, there is a considerable loss of

throughput to the tune of 7% for long flows (in comparison to DCTCP), and the throughput loss increases with the offered load.

## 2.3.2. Arbitration

In data center transport protocols [12], [47] employing arbitration transport strategy, switches make the scheduling decision, taking in to consideration all the network flows and their associated priorities (expressed in terms of flow size or deadline). The outcome of scheduling decision is conveyed to the end-hosts, as a rate at which flows should send data. In order to pause a low priority flow, the rate could be set to zero on the other hand it could be set to full capacity for high priority flows. [12], [47] shows that, though arbitrations is a centralized problem in general, it could be implemented in a decentralized manner. This would require each switch in a flow's path to add its rate to the packet header and the sender (end-host) to select the minimum rate for sending data.

Arbitration based transport protocols, by virtue of their explicit nature, helps the flows to attain their desired rate quickly. In addition, these schemes facilitates strict priority scheduling of flows by allowing flow with highest priority to utilize the full link capacity by pausing lower priority flows. There are also a number of problems associated with the explicit rate assignment based schemes. Important among them are listed here. Determining accurate rates for flows is quite hard, considering the possibility of flows getting bottlenecked at non-network resources like source application or at the receiver. Flow switching overhead, associated with pausing and unpausing flows is another important issue. Flow switching overhead could be of the order of 1 to 2 RTTs, and carry significant weightage taking short flows and high loads(which would require frequent pre-emption of flows) in to consideration.

### 2.3.2.1. D3

Deadline Driven Delivery Protocol: D3 [47] is the first research proposal that explored the idea of incorporating deadline information, for flow prioritization in data center networking. D3 is a clean slate approach, and the key idea behind the control protocol is to use flow deadlines associated with data center flows to schedule flows that are about to hit their deadlines over others that have their deadlines at a later point of time. D3 employs explicit rate control, to reserve bandwidth in accordance to their flow deadlines, and is based on the

assumption that the flow size and deadline information are available at flow initiation time. D3 uses deadline information to regulate the rate at which end-hosts inject traffic in to the network. End-hosts use flow deadline and size information, exposed by applications at flow initiation time, to request rates from switches along the data path to the destination. Switches, then apportion rates to flows to greedily fulfil as many deadlines as feasible. D3 tries to maximize application throughput, while ensuring burst tolerance, and high utilization in the data center network. D3 being a clean slate approach which tries to align the data center network to application requirements, requires changes to applications, end-hosts, and network elements. This aspect bodes as the biggest hurdle in D3 deployment. Though D3 reduces the deadline miss ratio in comparison with DCTCP, its operation involves significant control overhead in contrast to DCTCP. Another deterrent is the inability of D3 to co-exist with existing data center networking protocols viz TCP, DCTCP etc. preventing incremental deployment and thereby requiring a flag day where the entire data center switches over to D3 for it to be deployed.

#### 2.3.2.2. PDQ

Pre-emptive Distributed Quick flow scheduling: PDQ [12] is designed to facilitate flows to complete quickly and meet their deadlines. PDQ proposes a lightweight distributed flow scheduling layer for data centers, which could schedule and operate in Earliest Deadline First (EDF) and Shortest Job First (SJF) mode, using only FIFO tail drop queues. PDQ can reduce average flow completion time by 30% compared to D3 and TCP. PDQ requires a shim layers to be introduced between transport and network layers and also requires changes to the network fabric's hardware and software.

### 2.3.3. In-network Prioritization

Data center transport protocols that are based on the in-network prioritization strategy ([48]), requires packets to carry flow priorities. In such a scheme, flow priorities could indicate the flow deadline or size, and switches makes decisions based on this priority to schedule or drop packets (in case of congestion). In-network prioritization strategy helps to achieve work conservation - by scheduling a lower priority packet in the absence of higher

priority packets, and pre-emption – by ensuring that a higher priority packet on its arrival gets precedence over a lower priority packet.

Most significant hindrance to the adoption of in-network prioritization based strategy is the limited availability of priority queues in switches, which typically ranges from 4 to 10 [43]. Considering the number of unique flow priorities in the system in a practical setting, the number of queues is much smaller than required. In multi-link, scenarios local decisions about prioritization taken by switches could result in sub-optimal performance, which is another concern about this strategy [43].

#### 2.3.3.1. HULL

High-bandwidth Ultra-Low Latency: HULL [39] architecture aims to achieve near baseline fabric latency and high throughput at the same time, making use of the link utilization approaching its capacity as a means to detect and signal the onset of congestion. HULL caps link utilization at less than link capacity and makes use of Phantom Queues (virtual queues) associated to each egress switch port to ECN mark packets. Phantom queues drains at a lesser rate than the actual link's rate and ECN marking is done based on the phantom queue occupancy, which in turn corresponds to the link utilization. This way HULL tries to eliminate buffering and ensures almost zero queue occupancy, thereby allowing latency sensitive short flows to avoid the delay that could result from having to wait at the switch's egress queues. HULL uses DCTCP congestion control algorithm, to respond to the extent of congestion, and mitigates the bandwidth penalties that could result from operating in a buffer less manner. HULL also requires the support for packet pacing at the end-hosts to mitigate the adverse effect of packet bursts, common place in data center environment due to network features such as Large Send Offloading and Interrupt Coalescing, which could otherwise be interpreted as congestion signals by the phantom queues. Key principle that motivated HULL design is the idea to trade - bandwidth - a resource that is comparatively abundant in modern data centers, for - buffer space – the resource which is expensive to deploy and could contribute to significant increase in latency. Evaluation results presented in [39] show that HULL could reduce average and 99th percentile packet latency by a factor of 10 compared to DCTCP and a factor of 40 compared to TCP with 10% reduction in effective bandwidth as a trade-off. HULL requires extensive changes in the end-hosts for supporting

packet pacing and also requires changes in in-network switches for phantom queues which are not trivial to implement and deploy.

#### 2.3.3.2. pFabric

pFabric [48] is a clean slate data center transport, designed with the aim to attain next to optimal flow completion time for data center traffic flows. pFabric design is based on in-network prioritization and performs packet scheduling and dropping based on flow priorities. The key strategy employed by pFabric design is the decoupling of flow scheduling from rate control. This is achieved by associating a single priority to packets that belong to a specific flow and using switches that implement a priority queuing, as opposed to a FIFO queue. End-hosts mark each packet with a priority that could be based on deadline or flow's remaining size. For example short flows that are approaching their deadlines which are flagged to be critical to user experience are marked with high priority value in their packet headers. Switches in pFabric has very small buffers and makes greedy decision about packets to be accepted in to the buffer and the packets to be scheduled for transmission based on the packet's priority. Packets arriving at the switch during congestion are dropped if it has lower priority than all the packets in the buffer else it is accepted by dropping the lowest priority packet from the buffer. Packets enqueued in switch queues are sorted in the non-descending order of priority so that highest priority packet in the buffer is selected for transmission first. pFabric employs a bare-minimal rate control mechanism, where all flows commence transmission at line rate and reduce their sending rate only at the onset of high and persistent loss. pFabric design requires implementing a clean-slate network stack and assumes that applications supplies flow priorities to the transport layer protocol up on flow initiation. Another aspect that impedes the deployment of pFabric is that, it requires support for a large number of priority levels which is not available in current day network switching fabric and thus warrant significant changes the network fabric [43].

### 2.3.4. Multipath Data Center Transport Protocols

There are also recent research proposals [16], [40], [49], that try to exploit multipath nature of data center network in devising transport strategy based on load balancing, to achieve better latency and throughput. Though this thesis research work focuses on the single path

based transport protocols, a brief on the major multipath data center transport protocols are given below for ensuring completeness.

DeTail [24] proposes a clean slate approach that makes use of cross-layer, in-network mechanisms to prioritize latency-sensitive flows, and evenly balance traffic across multiple paths, thereby reducing the long tail of flow completion times. DeTail is shown to reduce 99.9th percentile flow completion times by over 50% in comparison to DCTCP. DeTail requires custom switch fabric and also requires changes to the entire network stack starting from physical layer all the way up to application layer.

MPTCP [49] is an end-host driven dynamic load balancing scheme, that splits a TCP flow into multiple sub-flows over a number of MPTCP ports, there by trying to avoid flow collisions and hence reduce congestion. MPTCP sender monitors packet loss, achieved throughput and RTT across the sub-flows and imparts flow prioritization. MPTCP incur additional overhead at the receiver to stich the multiple sub-flows together. MPTCP requires more than 10,000LOC kernel code changes, due to complex sender side congestion control algorithm and the reassembly code at the receiver. Though MPTCP is shown to achieve better performance for flow sizes greater than 70KB; it adversely affect flow completion time for mice (short) flows with sizes less than 70KB [50]. MPTCP incurs higher CPU overhead to the tune of 40% in MPTCP clients and 10% on MPTCP servers [49], which is considered expensive in a data center setting.

Explicit Multipath Congestion Control Protocol (XMP) [16] proposes mechanisms to improve the MPTCP's congestion control scheme (which was designed for transport characteristics of Internet). MPTCP congestion control fills up link buffer in order to attain full link utilization thus increasing the probability for packet loss due to queue build up and hence adversely affect short flows. XMP improves MPTCP congestion control by incorporating trade-off between latency and throughput, which is a prominent characteristic of data center traffic. In order to attain the same, XMP uses two algorithms, Buffer Occupancy Suppression algorithm - that helps to control the link buffer occupancy there by helping short flows to attain better latency and Traffic Shifting algorithm - which moves traffic corresponding to an MPTCP flow from congested paths to less congested ones, thereby helping long flows to achieve better throughput compared to MPTCP.

## 2.4. Discussion

Schemes DCTCP, ECN*, HULL try to achieve lower latencies for short flows by reducing the time short flows have to wait at the egress queues. Schemes focused on ensuring low switch buffer occupancies could benefit from short flow prioritization as shown by the deadline aware transport protocols D3, D2TCP, PDQ and size aware protocols L2DCT. Deadline, size aware protocols are designed under the assumption that applications passes on details about the deadline or size of the flows to the transport layer, which is not the case in the current data center environment. PDQ and D3 make use of explicit rate assignment based on estimated flow completion time and deadline. Though these approaches could potentially provide very good performance, challenges and the complexity associated with implementing them in practice act as deterrents in their adoption. In order to perform flow scheduling these schemes requires the network to explicitly assign a rate to each flow. This requires maintaining flow information (desired rate, size, RTT, deadline etc) at switches and coordination among switches to identify the bottleneck for each flow. In the highly dynamic data center environment this accounts for a major burden, both in terms of state maintenance and communication overhead at the switches. Clean slate mechanisms like De-Tail and pFabric require changes in the networking elements which limits their deployability and adoption.

This is where the data center transport schemes proposed in this paper assumes significance by virtue of being deployment friendly and at the same time achieving significant improvement in flow completion time for short flows and throughput for long flows. This thesis's first proposal, FA-DCTCP, taps in to the information readily available at the transport layer (IP, port number), and the ability to monitor TCP sockets, to identify the nature of the flows (short or long) and leverage this to provide differential treatment to short and long flows. In essence FA-DCTCP, based on the self-adjusting endpoints strategy, proposes differential treatment of short and long flows by incorporating flow awareness in to DCTCP. Evaluation results show that by doing so, it is possible to achieve better FCT for short flows at the same time not compromising on the throughput for long flows. The second contribution from the thesis research, extends ECN* and DCTCP, using a SDN based framework to achieve better long flow throughput. This again is a deployment friendly solution that does

not require any changes in the network elements and could be deployed as a simple application running on the SDN controller in a SDN based data center network.

## 2.5. Software Defined Networking

Software Defined Networking (SDN) attempts to bring in better flexibility for control application development and deployment in computer networks, a large scale distributed system. Typical SDN architecture has three decoupled layers – control application, control plane and data plane. SDN control plane translates requests from the control applications to the data plane forwarding elements and provide control applications with an abstract view of the network constituted by the forwarding elements.

Network technology has evolved rapidly over the years in terms of dramatic growth in link capacities, port densities and performance to cost ratios. At the same time control plane mechanisms, protocols where intelligence to control and manage the network resides were not able to evolve that fast mainly due to the lack of network wide abstractions that could shield control plane protocol design from low-level details. Another side effect of this is that computer networks becoming more complex to manage in fact it takes an order of magnitude more sysadmin effort to administer a network node than a single computational node. This again stems from the fact that the lack of abstractions in the control plane has led to a scenario where collection of complex protocols reside in the control plane as opposed to simple protocols that the early Internet started off with.

Figure 4 depicts the traditional network, highlighting the fact that an individual network node is composed of data plane and control plane. Data plane operates at nano-second time scale processing packets based on local forwarding state. Data plane is able to evolve rapidly and accommodate new technologies because of the layered abstractions, provided by the Internet TCP/IP stack. Applications built on reliable/unreliable transport which is built on best effort global packet delivery which is built on best effort local packet delivery built on local physical transfer of bits. This has made the Internet the widely adopted communication infrastructure. Control Plane operates at the scale of 10s of millisecond populating the forwarding state based on manual configuration and control protocols implementing distributed algorithms. Control plane deals with inherently non-local information and

currently does not have any abstraction to make the life easy for control protocols. As a result control protocols has to often re-invent the wheel towards gathering state information, node discovery and ultimately drawing up the network topology before applying the intended logic on it. Classic example is the OSPF protocol which implements mechanisms on its own to figure out the network topology by exchanging messages with OSPF protocol running on neighboring nodes. After figuring out the network topology OSPF protocol applies the Dijkstra's algorithm to compute the shortest paths. OSPF RFC has 255 pages out of which 250 pages talks about the mechanisms to figure out the network topology and 5 pages talks about the Dijkstra's algorithm.

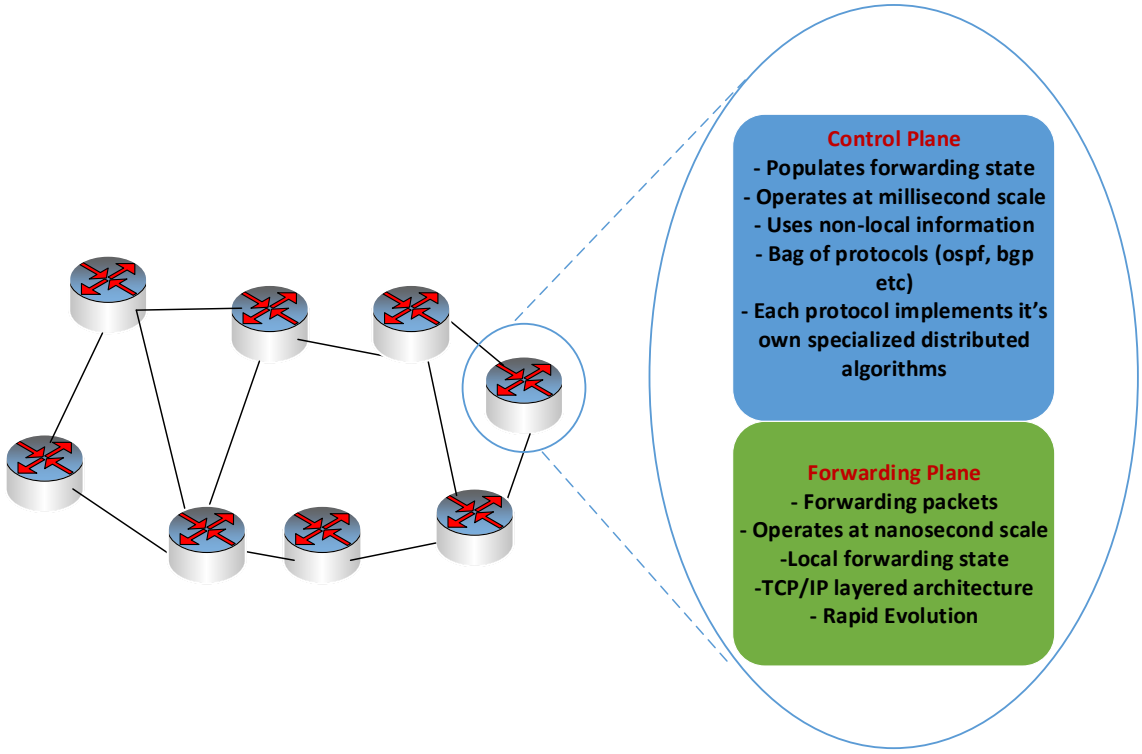


Figure 4 : Traditional Network

Software Defined Networking is an attempt to address the above mentioned concerns by introducing a paradigm where (a) the control plane and data plane are decoupled (b) control applications could be developed on top of abstractions exposed by control plane. Typical SDN architecture (depicted in Figure 5) has three decoupled layers - control applications, control plane (controller) and data plane (forwarding elements). Control applications are

programs that observe and implement the desired network behaviour by interfacing with the controller. Control applications make use of the abstract view of the network provided by controller for their internal decision making purposes. SDN controller translates requests from the control applications to the data plane forwarding elements and provides control applications with an abstract view (including network state, statistics and events) of the network constituted by the forwarding elements. SDN controller takes the role of network operating system which will in effect abstract away the need for dealing with underlying network details from the control application; rather they can operate on the view of the network provided by the controller. The key concept here is that the controller - a distributed system that creates a consistent, up-to-date global network view - acts as an abstraction layer, sits in between the data plane and control applications and shield control protocol design from low-level details.

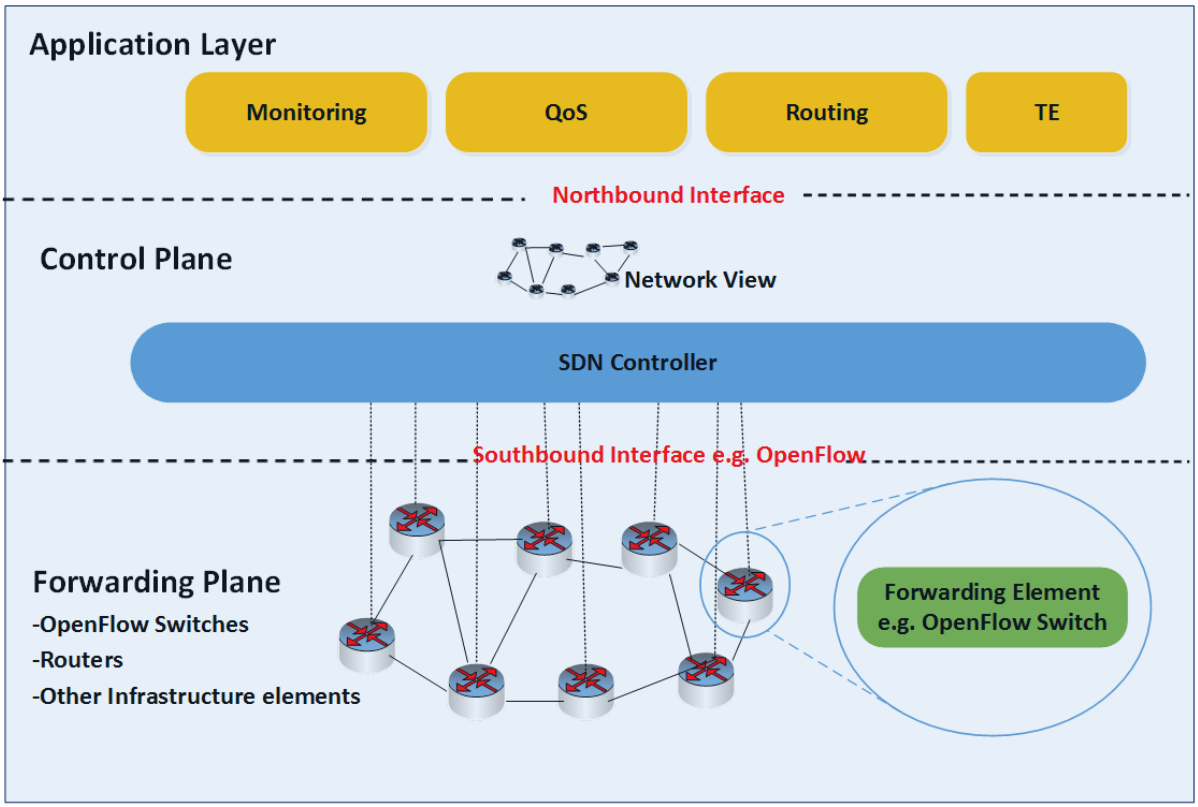


Figure 5 : Software Defined Network architecture

SDN exposes set of standard open APIs between controller and forwarding elements (southbound APIs e.g. OpenFlow) and APIs between controller and control applications (northbound APIs, standardizations efforts are in progress). Southbound interface is an open,

vendor neutral interface between Controller and Forwarding elements. This interface facilitates (a) programmatic control of forwarding elements and operations (b) advertisement of capabilities (c) reporting of statistics and (d) notification of events. Northbound interface provides Open APIs between the SDN control and applications layers. Northbound interface allows control applications to operate on an abstraction of the network, leveraging network services and capabilities without being tied to the details of their implementation. By virtue of exposing the network view abstraction to the control application northbound interface facilitates direct expression of network behaviour and requirements [51].

SDN helps to greatly simplify the network devices (forwarding elements) since they no longer need to understand and process hundreds of protocol standards but merely accept instructions from the SDN controllers. Controller uses a forwarding plane abstraction configuring flow tables, forwarding rules, etc in the forwarding elements. SDN frees the network forwarding elements from being "application-aware" and control applications from being "network-aware" [52].

**Software Defined Networks in Data Centers:** Ability offered by the SDN paradigm to develop and deploy custom network applications and protocols as applications running on the SDN controller has triggered the adoption of SDN in data centers. Research proposals Hedera [27], PortLand [15], MicroTE [18], Applying NOX to the Data Center [53], Lime [54], OpenTCP [55], Mahout [56] etc. highlight this trend. Chapter 4 propose a SDN based mechanism to achieve better throughput for long flows that take advantage of the unique capabilities SDN brings in to the data center networking such as, (a) the ability to observe the network from a central vantage point (b) the ability to dynamically update network element configurations depending on the network state, and (c) the ability to deploy custom application on the SDN controller without requiring any changes in network elements.

# Chapter 3 - Improving Flow Completion Time for Short Flows

This chapter proposes a mechanism to improve Flow Completion Time for latency sensitive short flows in data centers. Proposed scheme belongs to the *Self-Adjusting Endpoints* data center transport strategy. The scheme extends DCTCP by incorporating flow awareness in to the congestion control mechanism and is named Flow Aware DCTCP (FA-DCTCP). DCTCP uses ECN based congestion estimation to scale the congestion window at the onset of congestion thereby reacting to the extent of congestion. This thesis work shows that it is possible to go a step further, incorporating flow awareness in to DCTCP. Flow awareness could be used to adapt the congestion algorithm in DCTCP, for meeting low latency requirement of short flows in a much better fashion.

High performance data center transport mechanism is often characterized as one which tries to achieve at least one of the following while not compromising on others - minimizing flow completion time, maximizing throughput or reducing deadline miss rate [11], [13]. FA-DCTCP focuses on minimizing flow completion time for short flows.

The remainder of the chapter is organized as follows: Section 3.1 discusses the FA-DCTCP design principles. Section 3.2 describes the FA-DCTCP architecture. Section 3.3 lists the details of FA-DCTCP operation. Section 3.4 covers the FA-DCTCP implementation details. Section 3.5 lists the conditions and constraints for FA-DCTCP. Section 3.6 covers the experimental setup and the evaluation results. Section 3.7 concludes the chapter.

## 3.1. Design Principles

This thesis introduces FA-DCTCP, a *self-Adjusting endpoint* data center transport mechanism, which leverages the end-host's ability to distinguish (determine) flow types. FA-DCTCP is built on top of the DCTCP and extends DCTCP control algorithm (mechanism) at the DCTCP sender. In this section, the proposed mechanism's design decisions are presented.

**Deployment friendliness:** Ease of deployment is a key factor that aids faster adoption of data center transport protocols [11], [43], [46]. FA-DCTCP requires changes only in end-hosts in the data center network. FA-DCTCP does not require any software or hardware changes in the in-network switches. FA-DCTCP design ensures deployment friendliness by requiring only a mere, simple, upgrade of TCP stacks in end-hosts in data center. FA-DCTCP design ensures that it is easier to implement and deploy with minimal changes at the end-hosts and without requiring any changes to the network fabric. FA-DCTCP requires only 52 lines of code change to the fast recovery phase of the congestion control algorithm employed by DCTCP sender and do not require any change to DCTCP receiver and ToR switch. FA-DCTCP meets the requirements for a deployment friendly data center transport scheme, by virtue of not requiring any hardware or software changes to the network fabric [43], [46].

**Leverage flexibilities offered by the Data Center Network Environment**: FA-DCTCP, similar to DCTCP, is tailored for the data center environment and deals with traffic internal to the data center. Characteristics of data center environment differ significantly from the wide area networks. Especially the network will be under single administrative control and will be mostly homogeneous. Single administrative domain nature of data center offers certain luxuries for designing a data center transport mechanism. The most prominent is the ability to deploy incremental OS updates on to the end-hosts in data center network [56]. FA-DCTCP design takes advantage of this fact for ensuring deployment friendliness. Flow type identification mechanism employed by FA-DCTCP leverages the insights in to well-known application patterns within the data center and the ability to map them in to unique five tuple at the transport layer, again a characteristic of the data center environment [19], [56], [57].

## 3.2. FADCTP Architecture

FA-DCTCP framework is proposed as an extension to the DCTCP component of the TCP/IP stack. Proposed FA-DCTCP mechanism categorises TCP flows into short and long flows and applies differential modulation of congestion window based on the flow type. Figure 6 presents the architecture of FA-DCTCP. As depicted, the proposed FA-DCTCP architecture has two main components - Flow classification module and Congestion control

module. Architectural details about the FA-DCTCP modules are given below. The inner workings of the modules are covered as part of the section outlining FA-DCTCP operation.

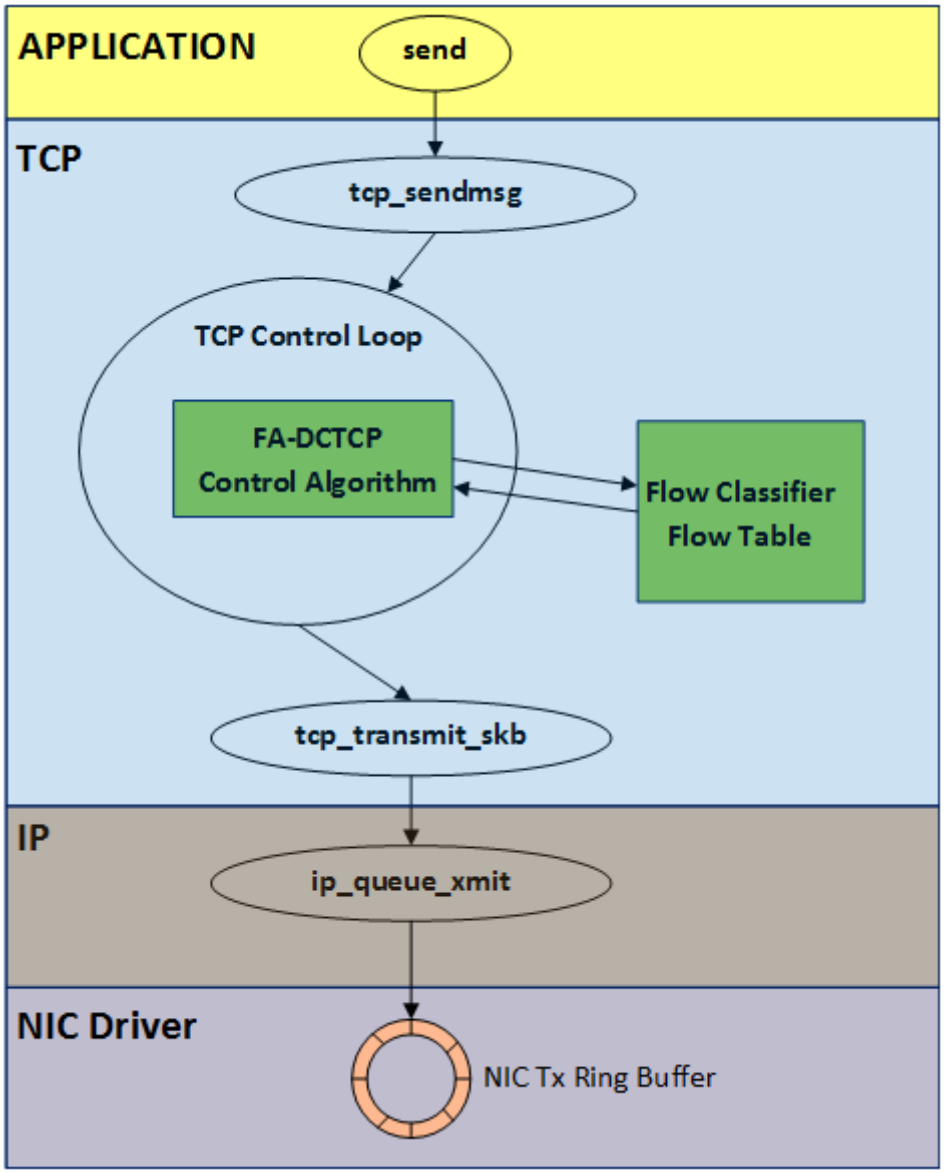


Figure 6 : FA-DCTCP Architecture

(a) **Flow classification module** performs the following functionalities.

1) Collects information required to classify TCP flows, in to short and long flows.
2) Populates a **flow table** where flow classification information is maintained.

(b) **FA-DCTCP congestion control algorithm (module)** influences the behaviour of TCP's fast recovery phase, by performing flow aware modulation of the congestion window during the fast recovery phase. Congestion control module retrieves flow type information from the **flow table** maintained by the flow classifier module.

Figure 6 illustrates the FA-DCTCP architecture in terms of functionality added to the (Linux kernel) TCP/IP stack. FA-DCTCP extensions are shaded in green. FA-DCTCP extends the sender side code in the networking stack with two simple changes, inserting only 52LOC in the TCP stack. The first change, in the TCP stack, is to add flow classification mechanism which classifies flows based on the five tuple and by monitoring socket buffers to identify the amount data transferred over a TCP connection. The second change is to invoke the FA-DCTCP control algorithm while in the fast recovery phase. FA-DCTCP control algorithm processing is initiated by a simple function call with in the congestion window calculation in the fast recovery phase.

## 3.3. FA-DCTCP Operations

FA-DCTCP builds on DCTCP, a TCP variant for data center networks designed to improve the flow completion time by maintaining low queue lengths in the switches. DCTCP design treats short flows and long flows equally, whereas the proposed enhancement – FA-DCTCP, aims to incorporate flow prioritization into DCTCP, by adding the ability to treat short flows and long flows differently. Essence of the idea behind FA-DCTCP is to; reduce the congestion window for short flows gracefully compared to long flows, thereby improving flow completion time for short flows. In order to guarantee that the throughputs of long flows are not affected congestion windows for long flows are reduced at the same rate as DCTCP. In short in the presence of congestion long flows back off aggressively, whereas the short flows back off gracefully in comparison to longs flows allowing them to complete faster. FA-DCTCP changes are limited to TCP stack in end-hosts and do not require any software or hardware changes in the switches. FA-DCTCP requires only a configuration change (ECN settings) in the switches. This makes deployment of the proposed scheme fairly straight forward requiring only upgrades to the TCP stacks at the end-hosts.

The main aspects that need to be taken care in FA-DCTCP operation are: (1) identifying short and long flows at the transport protocol layer of end-hosts (servers) in a data center, (2) designing a congestion control algorithm for adjusting congestion windows for short and long flows differently at the sender so that short flows complete faster compared to a deployment based on DCTCP. Specifics on how FA-DCTCP addresses these aspects are discussed below.

### 3.3.1. Identifying Short and Long Flows at the End-hosts

This could be done in either of the two ways listed below:

*(a) Distinguishing flows using the TCP port numbers corresponding to applications generating elephant flows:* This approach is based on the luxuries a data center environment provides in terms of being a single administrative domain and the ability to take advantage of application awareness as pointed out in [19], [56]–[59]. Data center being a single administrative domain facilitates employment of schemes that could determine the type of a flow *a priori* by looking at the application port number. Martin Casado et al. [58] points out that elephant flows can be determined *a priori* without actually trying to detect them from network effects. This is since as an artifact of data center environment design, elephant flows are either related to VM cloning, backup, data-structure refresh, file transfer, or VM migrations, all of which can be identified from the edge (at the end-hosts) looking at the TCP headers. Wu et al. [57] and Abts et al. [19] proposes the usage of canonical five-tuple, comprising of source/destination IP address, source/destination port and protocol number, to uniquely identify data center transport flows. Another work justifying this approach is the Google's B4 architecture [59] which identifies the elephants and mice flows *a priori* before applying specific traffic engineering rules pertaining to them.

*(b) Monitoring TCP socket buffer per TCP connection:* This approach is based on the strategy proposed in Mahout [56] to distinguish between mice and elephants flows. Idea here is to identify elephant flows by setting a threshold on the TCP socket buffer usage per TCP connection. For instance FA-DCTCP implementation uses an approach where flows whose TCP socket buffer usage exceeds the threshold set at 1MB are marked and flagged as elephant flows. Threshold of 1MB is selected based on the data center traffic measurement

studies [2]–[4], [11] that had pointed out that mice flows range up to sizes 1MB and elephants 1MB to 1GB.

Flow classification module and the associated flow table, shown in Figure 6, performs the job of deciphering short and long flows at the end-hosts. Flow classifier identifies the long flows by employing the strategies listed above. Flow classifier in-turn populates the flow table with the list of long flows in the data center. Flow table maintains the key data structure in the FA-DCTCP design which stores the per flow information for all the active long flows. A long flow is uniquely identified by a five-tuple: source/destination IP address, source/destination port and protocol.

### 3.3.2. FA-DCTCP Control Algorithm

Flow aware DCTCP congestion control algorithm is built on top of DCTCP by adding flow awareness to it. Switches in this setting are ECN configured to mark packets when the buffer occupancy reaches a stipulated marking threshold, $K$. An exponentially weighted moving average of the level of congestion, denoted as $\alpha$, is maintained at the sender. $\alpha$, gives an estimate of the fraction of packets marked, updated once for every window of data.

$$\alpha = (1-g) \times \alpha + g \times F \qquad (1)$$

$F$, indicates the fraction of packets marked in the last window, and is the measure of packets that encountered congestion during that period. $g$, a real number between 0 and 1, is the weight given to new samples compared to previous ones in the calculation of $\alpha$. Based on $\alpha$, congestion window $cwnd$ is resized as follows:

$$cwnd = cwnd \times (1- \alpha^{\beta}/2) \qquad (2)$$

where $\beta$ is the factor used to incorporate flow awareness in to the congestion algorithm, and is used to modulate the congestion window differently for short and long flows. $\beta$ is set to 1 for long flows; ensuring congestion window for long flows are scaled in the same fashion as in DCTCP. In the case of short flows $\beta$ is set to 2, during high levels of congestion ($\alpha > 0.6$), and is set to 3 otherwise ($\alpha \leq 0.6$). This way the congestion windows for short flows are scaled down at a lower rate in comparison to long flows.

FA-DCTCP control algorithm is presented below:

---

**Algorithm 1: FA-DCTCP congestion window calculation**

---

*CurrentCwnd: Current value of congestion window*

*Alpha: Fraction of the packets that are ECN marked*

*NewCwnd: Congestion window calculated by the algorithm*

**Input:** < CurrentCwnd, Alpha >

**Output:** < NewCwnd >

// if flow is of type long (i.e. an elephant flow)

**if** FLOW_TYPE == ELEPHANT **then**

//calculate CWND in the same fashion as DCTCP, using //DCTCP decrease law

CwndNew = CurrentCwnd × $(1 - \frac{\text{Alpha}}{2})$

**else** // if flow is of type short (i.e. a mice flow) calculate

//CWND using FA-DCTCP decrease law

**if** (Alpha > .6) **then**

//during high levels of congestion, reduce congestion

//window at a higher rate

CwndNew = CurrentCwnd × $(1 - \frac{(\text{Alpha})^2}{2})$

**else**

//during low levels of congestion, reduce congestion

// window at a lower rate

CwndNew = CurrentCwnd × $(1 - \frac{(\text{Alpha})^3}{2})$

**end if**

**end if**

---

### 3.3.3. FA-DCTCP Congestion Window Scaling Factor

This section details the choice of $\beta$, and therefore the congestion window scaling factor ($\alpha^{\beta}$), used in the algorithm. Congestion window scaling factor ($\alpha^{\beta}$) influences the behaviour of the FA-DCTCP algorithm, in terms of the ability to achieve better performance (measured in-terms of improved flow completion time), while ensuring it does not lead to congestion collapse (indicated by queue over flow).

Choice of $\beta$ was based on experimental evaluations where the objectives were to satisfy the following three criteria:

a) Short flows should not drive network to congestion collapse by virtue of reducing congestion windows sparsely.
b) At the onset of extreme congestion, FA-DCTCP should cut congestion window for short flows aggressively (but still at a rate less than long flows) and should default to DCTCP as $\alpha$ approaches 1.
c) Should achieve significant Flow Completion Time improvement (above 10%) for short flows.

Experimental evaluations were conducted for various values of $\beta$, ranging from 2 to 6, as per guidelines outlined in [46]. Criteria (a) and (b) is measured in terms of queue overshoot frequency. Criterion (c) was evaluated by comparing the FCT. Figure 7 plots the congestion window scaling factor $(\alpha^{\beta})$ selected and used by FA-DCTCP for short and long flows for possible values of $\alpha$. Straight line in the middle of the plot, which corresponds to $\beta$ = 1, applies to long flows. Curve below the straight line applies to short flows. The portion of curve where $\beta$ = 3 applies to short flows during mild congestion and the portion corresponding to $\beta$ = 2 applies to short flows during higher levels of congestion. During mild and minor level of congestions both long flows and short flows reduces their congestion window slowly. In comparison, rate of reduction for short flows are lower than that for long flows by a power of three allowing short flows to grab more bandwidth and complete faster. For higher levels of congestion, rate of reduction of congestion window for short flows are lower than that of long flows by a power of two. As $\alpha$ nears 1 congestion window scaling factor for both short and long flows converge to 1.

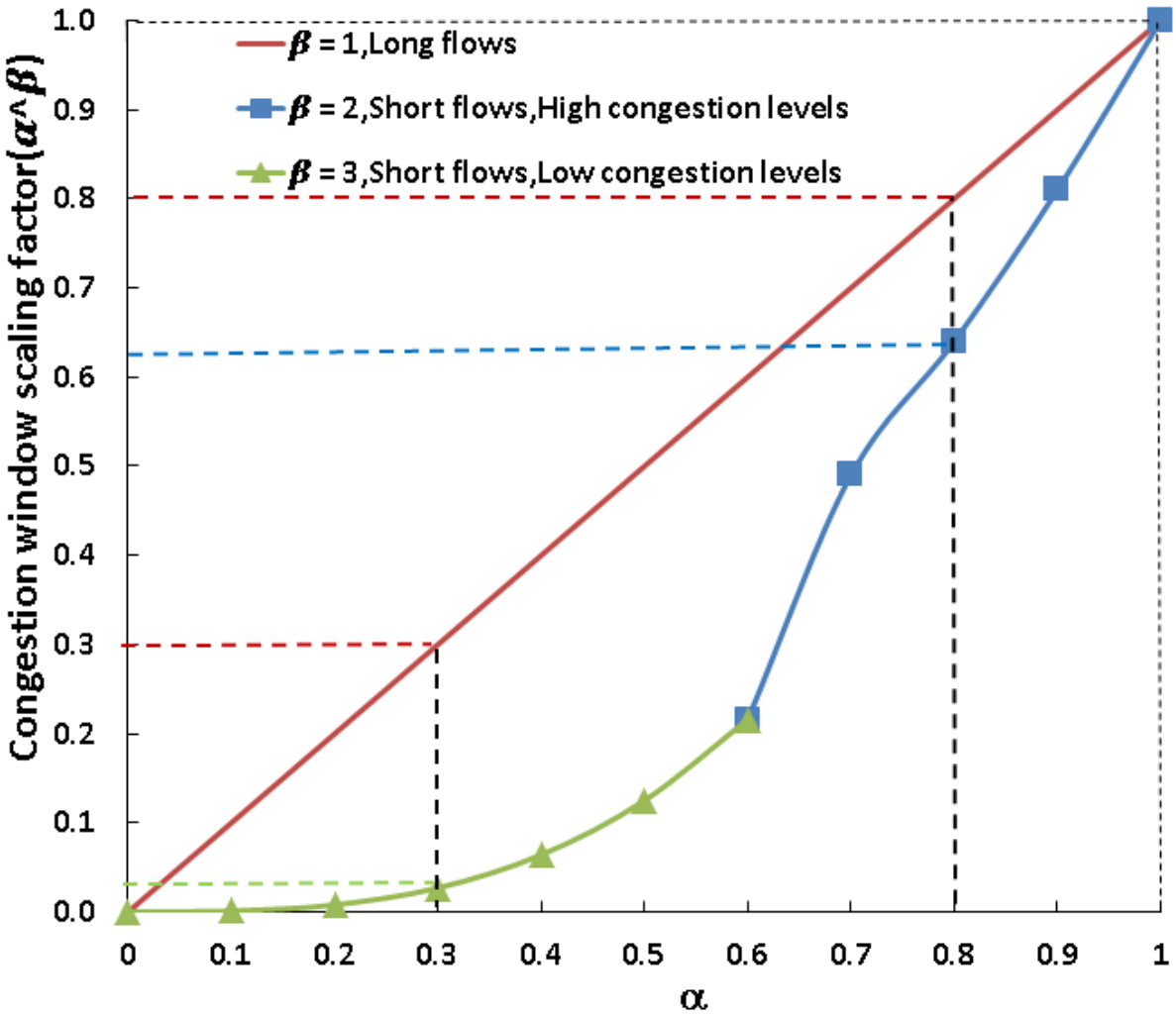


Figure 7: Plot of $\alpha$ vs Congestion window scaling factor ($\alpha^{\beta}$)

## 3.4. FA-DCTCP: Implementation Details

FA-DCTCP is implemented as a prototype, on top of the DCTCP Linux source code available at [60]. FA-DCTCP implementation is integrated into existing TCP stack, since it is an approach for TCP congestion window adaptation to achieve differential treatment of flows. FA-DCTP prototype is implemented as a lightweight plugin module that sits in the TCP stack and interfaces with the TCP congestion control mechanism. This required only 52 lines of code change to the TCP implementation in Linux kernel (version 3.2).

## 3.5. Conditions and Constrains for FA-DCTCP

FA-DCTCP, similar to DCTCP, is tailored for the data center environment and deals with traffic internal to the data center. In a data center environment connectivity to the external Internet is managed through application proxies and load balancers that separate internal traffic from external [11]. So scenarios of FA-DCTCP interaction with conventional TCP are not considered relevant in the purview of FA-DCTCP design and operation. Similar to DCTCP, FA-DCTCP algorithm comes to play only during the fast recovery phase; rest of the TCP operation like slow start, additive increase in congestion avoidance phase are retained as it is in original TCP implementation. Since DCTCP is single path in nature, FA-DCTCP also essentially operates in an environment where flows are uniquely identified by the canonical five tuple. Another assumption used in the FA-DCTCP design based on the literature [11], [12], [39], [40], [46], [47] on data center networking transport protocols is that, it is safe to assume that the short flows and long flows would not use the same TCP connection, i.e., they will not be interleaved over the same TCP connection in any case.

## 3.6. Evaluation and Analysis

This section captures the details regarding emulation based experimental tests and the criteria used to evaluate FA-DCTCP. Section also covers the results from the evaluation tests and the discussion about the same.

### 3.6.1. Experimental Setup

Details about the Mininet based experimental setup and the Iperf traffic generator used for evaluation of FA-DCTCP are captured here.

#### 3.6.1.1. Mininet based Experimental Network

Evaluation tests are performed using Mininet [61], a high fidelity network emulator that facilitates repeatable, realistic network experiments. Mininet is a Container Based Emulator and can be used to create a full-fledged virtual network, running real Linux kernel, on a single machine. For each virtual host in the test topology, Mininet creates a container

attached to a network namespace, which has a virtual network interface, and associated data. Virtual network interfaces are attached to software switches via virtual ethernet links.

Mininet facilitates rapid prototyping of large networks on the constrained resources of a single laptop. In-order to emulate a network, Mininet runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; and allows running arbitrary programs (including anything that is installed on the underlying Linux system.) Miniet allows writing programs that can emulate sending packets through a real Ethernet interface, with a given link speed and delay and the network elements like switch, router etc with configured amount of queueing. In a nutshell Mininet emulated networking elements (virtual hosts, switches, routers, links etc) behave akin to discrete hardware elements though they are created using software [62].

Mininet is well suited for experiments that are network-limited, i.e. class of experiments that are constrained by network properties such as bandwidth, latency, and queueing, rather than by other system properties namely memory access latency or disk bandwidth. Handigol et al. [61] points out that testing how a new version of TCP congestion control fares in a specific topology on 100 Mbps links would exemplify an excellent use case for Mininet, since the results from such a test will wholly be dependent on link bandwidth and latency. As proven in [61], Mininet is well suited for recreating the DCTCP baseline results. All these reasons contributed to selecting Mininet for validating characteristics of the proposed FA-DCTCP mechanism. Tests were performed using Mininet running on a Ubuntu LTS box with the following configuration - Intel Core i5 with two 2.9 GHz cores and 16 GB of RAM.

#### 3.6.1.2. Iperf Traffic Generator

The traffic generation for all the test scenarios is performed by Iperf [63]. It is a relatively simple traffic generation tool that is based on client-server architecture for measuring TCP throughput, flow completion time and other performance indicators. An Iperf client connects to an Iperf server and exchanges a few test parameters to conduct a performance

measurement. After which the bulk data transfer starts. During the tests, intermediate results can be displayed at configurable intervals.

The following command is an example of Iperf usage.

```
$ iperf -c 10.0.0.1 -p 5001 -n 500K -i .01 -E
```

The command will generate a TCP connection from host (h1) on which the command is executed to Iperf server (h2) running at the IP 10.0.0.1 (-c option) and listening on port 5001 (option -p). h1 starts to transfer 500KB bulk data (-n option) to h2. Inter arrival times between packets follows an exponential distribution (option -E). Iperf generates periodic bandwidth reports at intervals of .01 seconds (option -i).

### 3.6.1.3. Test Topology and Parameter Settings

Though a typical data center network topology is a 3-tier tree topology, in the evaluation test setup the core and aggregation layers are abstracted away. This is since; the objectives of evaluation tests are solely to check the performance of the proposed FA-DCTCP transport protocol over a bottleneck link traversed by short and long flows. Evaluation is done based on test topology depicted in Figure 8, a benchmark topology which is often used in evaluation of data center transport protocols, derived based on the DCTCP paper [11] and Handigol et al.'s work on reproducible network research [61]. As alluded to in DCTCP paper [11] and related data center networking literature ([10], [47]), the bottleneck in data center networks is the shallow buffered ToR switch. The test setup consists of a ToR switch connected to 11 servers, emulated using virtual hosts and a virtual switch in Mininet. The test topology helps to emulate the ToR switch with bottleneck link carrying traffic corresponding to (a) partition/aggregate application structure (e.g. web search), (b) distributed computation frameworks (MapReduce, Dryad), and (c) background flows (VM Migration, data-structure synchronization) in a typical data center environment for the evaluation purpose.

Ten servers Sn (n = 1 to 10) act as senders and R represents the receiver connected to a single 100 Mbps switch. Switch interface connected to server R acts as the bottleneck link. Link latencies are set to 225µs achieving an average round trip time (RTT) of 900 µs which is typical of data center network RTTs. Switch buffer size is set to 4MB to conform to the shallow buffered nature of ToR switches in data center. ECN is configured in Mininet via

Linux Traffic Control's Random Early Detect queuing discipline (RED qdisc). ECN marking threshold is set as 5 packets as per the guidelines outlined in [11]. Another key DCTCP parameter the weighted averaging factor g is set to 1/16 as recommended in [11].

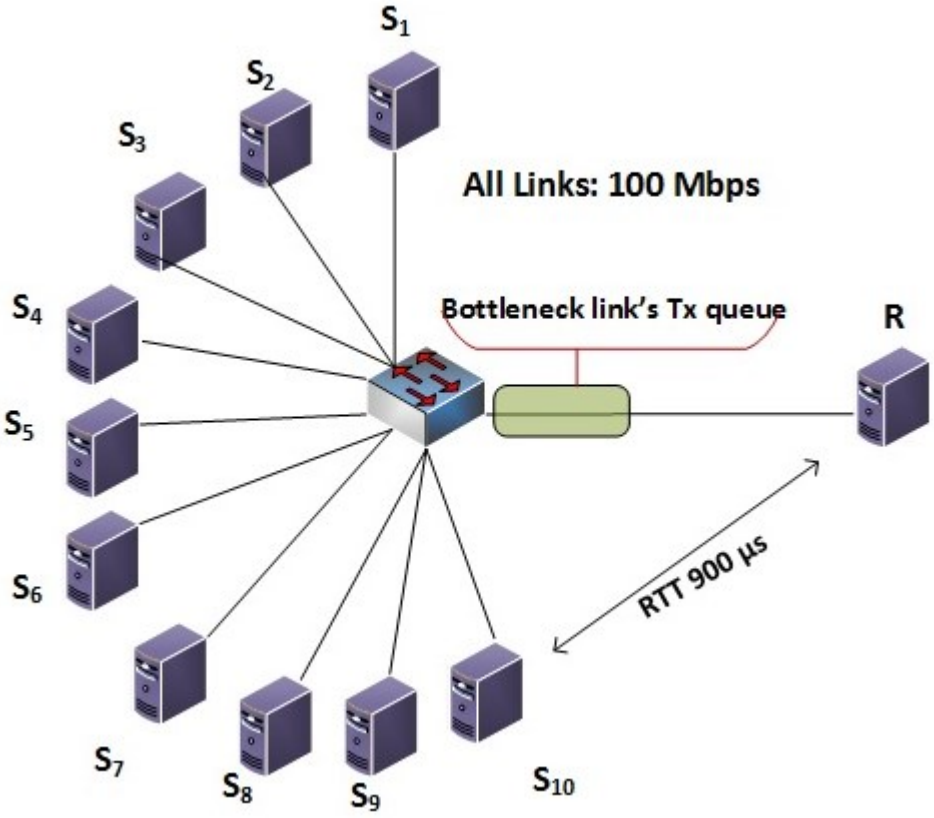


Figure 8 : Topology for Flow aware DCTCP evaluation experiments

Table 1 : Test setup parameter settings

| **Parameters** | **Value** |
|---|---|
| Link speed | 100Mbps |
| Link Latency | 225µs |
| Round Trip Time | 900µs |
| Switch Buffer Size | 4MB |
| ECN Marking Threshold | 5 Packets |
| Exponential averaging factor | 1/16 |

### 3.6.1.4. Test Workloads

Traffic workloads used for evaluation corresponds to the patterns - partition/aggregate, distributed computing and background traffic - observed in data center environments and is selected based on prior studies [2]–[4], [11], [12], [40]. Two long flows that span the test duration are originated from S1 and S2 towards R, corresponding to 75th percentile of multiplexing in data centers [11]. Each of the 10 senders originates short flows ensuring approximately 90 to 10 ratio of short flows to long flows as outlined in data center network measurement studies [2]–[4]. Inter arrival times of short flows follows exponential distribution and short flows sizes are selected from the set of values {250KB, 500KB, 1000KB} based on prior studies [11], [12], [40]. Test traffic is generated using Iperf traffic generator.

### 3.6.1.5. Test Setup Validation

In order to validate the test setup and to ensure that DCTCP behavior is recreated correctly in the test topology, two long flows are initiated from hosts S1 and S2 using Iperf traffic generator to the receiver R. Instantaneous queue occupancy at the bottleneck link is then monitored for 120 seconds, by sampling the queue length at every 0.4 second intervals. The observation from the test, depicted in Figure 9, is that queue length remains stable around 5 packets. This result compared against the result presented in Figure 6(b) from [61] and Figure 1 from [11] confirms the evaluation setup's correctness.

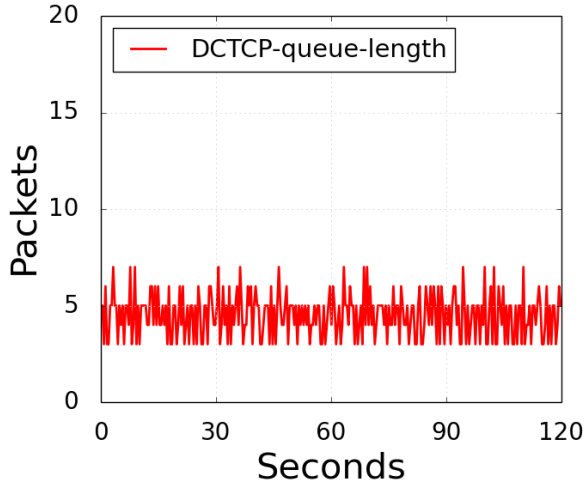


Figure 9: DCTCP instantaneous queue occupancy from the baseline test performed to validate evaluation framework

## 3.6.2. Evaluation Criteria

The objectives of the tests are to evaluate and compare the performance of FA-DCTCP to DCTCP. Metrics of interest in the evaluation, based on [11], are: (1) flow completion time (FCT) of short flows, (2) throughput of long flows, and (3) queue occupancy at the switch. To be acceptable, FCT for short flows should demonstrate a significant improvement, while ensuring that the throughput of long flows are not significantly affected and the queue occupancy remains low as in DCTCP. Tests were conducted with short flows of sizes 250KB, 500KB and 1MB. These tests were repeated with each algorithm, DCTCP and FA-DCTCP, with short flows of sizes 250KB, 500KB and 1MB, for 50 times. The results depicted are from a total of 150 test runs. Tests were long flow sizes were varied from 10MB to 1GB were also conducted to ascertain the proposed scheme's impact on long flow throughput. Results from the evaluation run are listed and discussed below.

## 3.6.3. Results and Discussion

This section discusses the results from FA-DCTCP evaluation tests.

### 3.6.3.1. Instantaneous Queue Occupancy

Figure 10 depicts the instantaneous queue occupancies at the bottleneck link, which remains stable around 5 packets, for both DCTCP and FA-DCTCP. This serves to exemplify the capability of FA-DCTCP to ensure low buffer occupancy similar to DCTCP.

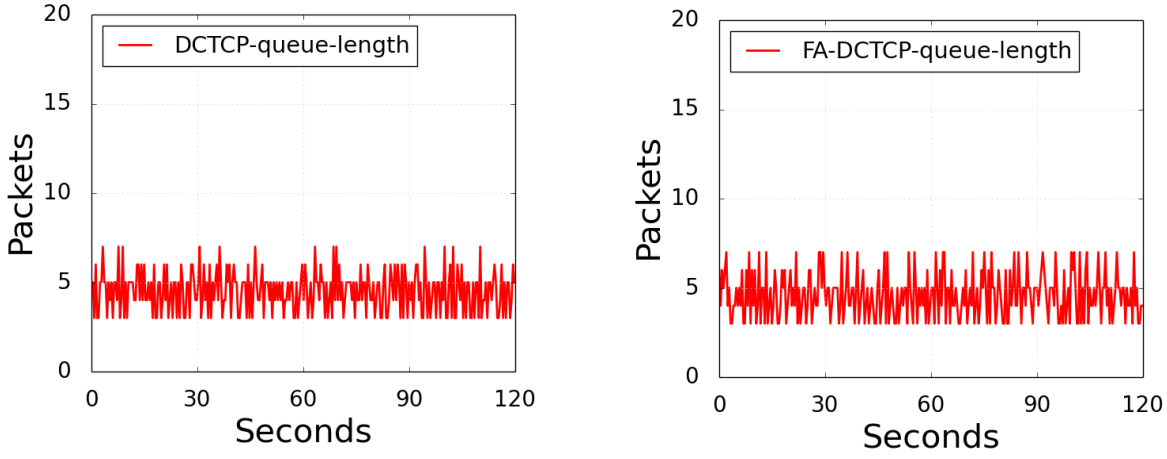


Figure 10: DCTCP and FA-DCTCP - Instantaneous Queue Occupancies

### 3.6.3.2. Average Flow Completion Time for Short Flows

Figure 11 captures the average flow completion time for 30 short flows. Tests were conducted for short flow sizes 250KB, 500KB and 1000KB, and each test were repeated 50 times. With FA-DCTCP, average flow completion time for short flows improves by 79ms which corresponds to an improvement (reduction) of up to 25.5% in average FCT is observed by the short flows. This significant reduction in FCT serves to highlight the usefulness of FA-DCTCP as good data center transport protocol.

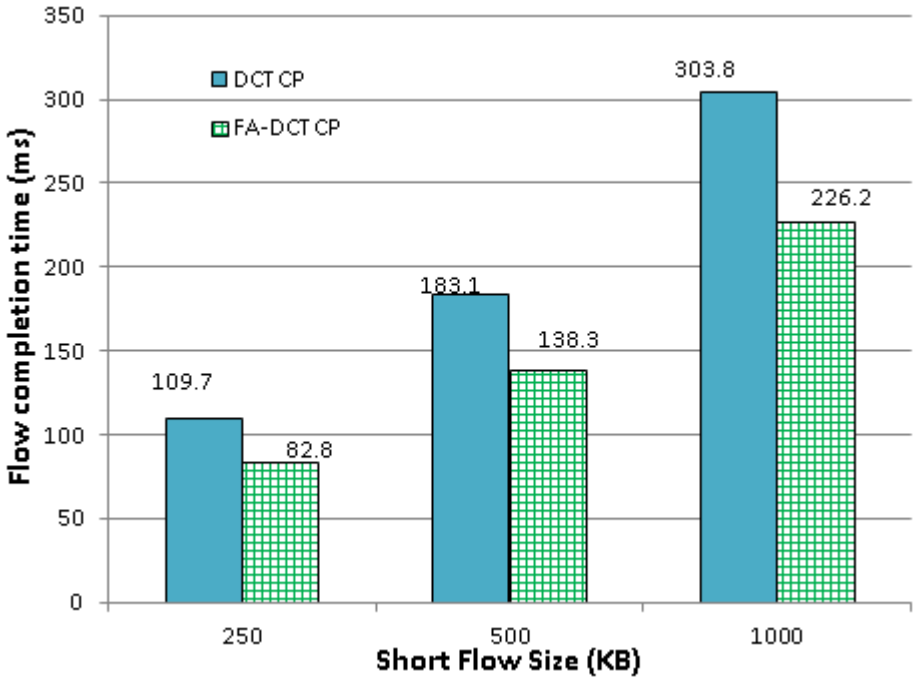


Figure 11 : Average flow completion time for short flows

### 3.6.3.3. 95th Percentile of Flow Completion Time for Short Flows

Figure 12 captures the 95th percentile of flow completion time for short flows. These are from tests conducted with 30 short flows per test and having short flows sizes varied to 250KB, 500KB and 1000KB across test runs. With FA-DCTCP, 95th percentile of flow completion time for short flows improves by 106ms which corresponds to an improvement (reduction) of up to 29.7%.

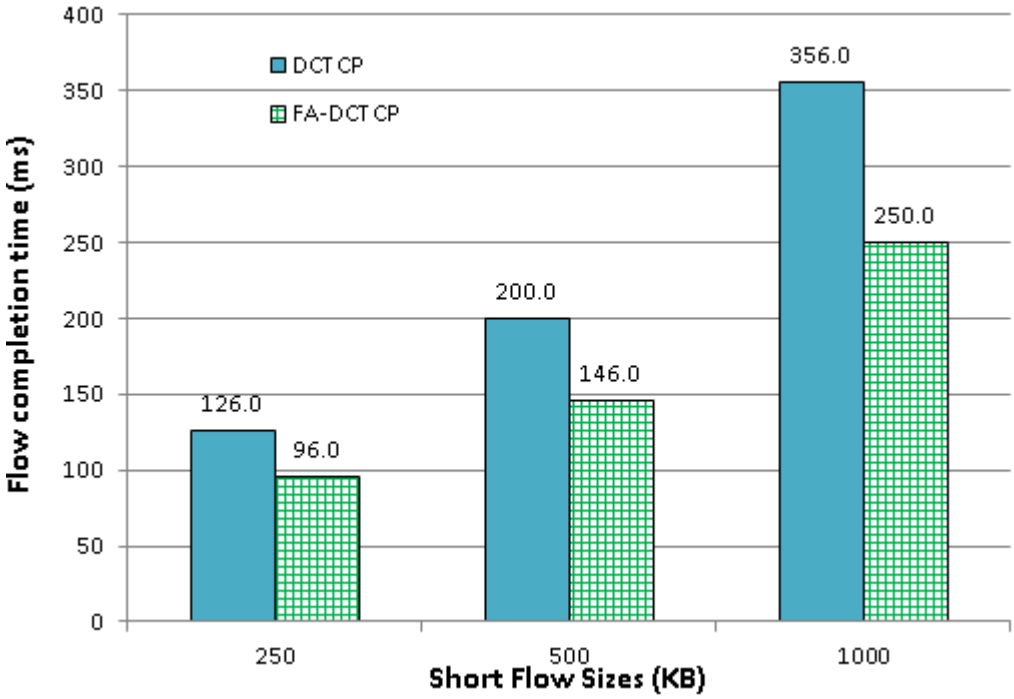


Figure 12 : 95th percentile of flow completion time for short flows

### 3.6.3.4. 99th Percentile of Flow Completion Time for Short Flows

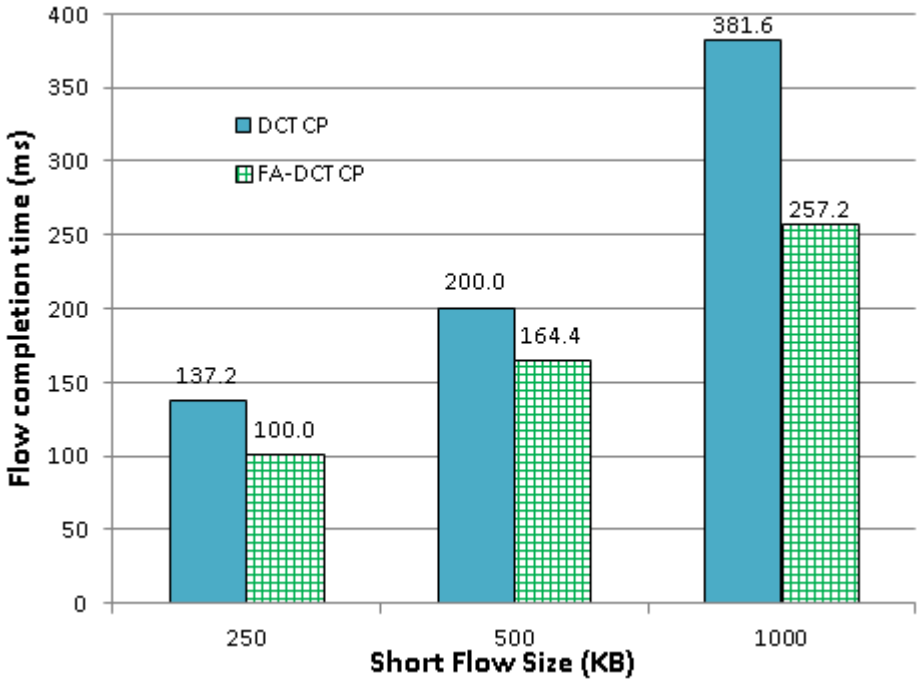


Figure 13 : 99th percentile of flow completion time for short flows

This metric corresponds to the tail latency in a cloud data center and is of much significance to applications like search, social networking page generation where the quality of page rendered depends majorly on the tail latency. As depicted in Figure 13, FA-DCTCP improves 99$^{th}$ percentile flow completion time by 124.4ms which corresponds to a reduction of 32.5% in the tail latency. As a result, FA-DCTCP reduces the 99th percentile of network latency for short flows employing partition/aggregate pattern by 32.5%. Thus FA-DCTCP frees up more time for computation in worker nodes and also reduces the latency observed by the end users.

### 3.6.3.5. Throughput of Long Flows

Throughput measurements for long flows from the evaluation tests, depicted in Figure 14, shows that FA-DCTCP introduces only a minor negligible reduction of 0.04% in long flow throughput.

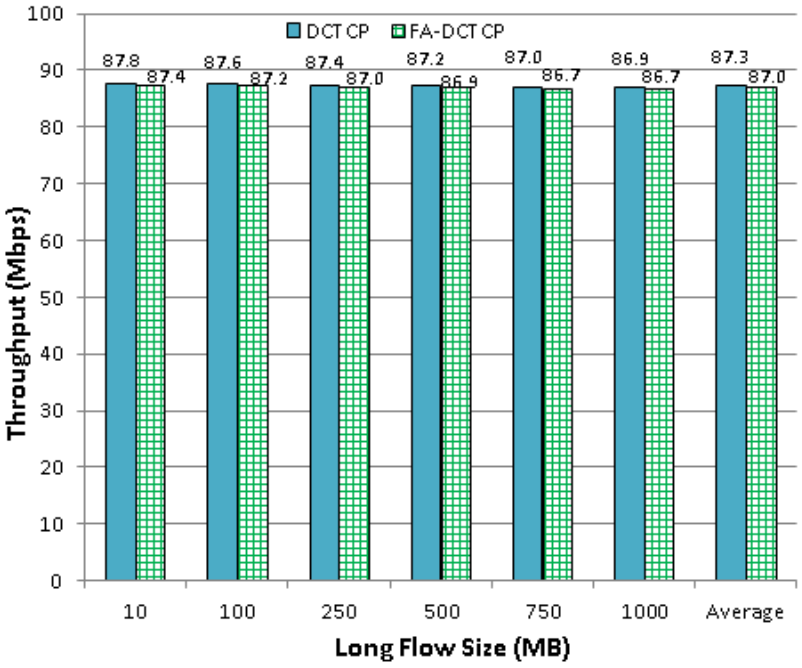


Figure 14 : Throughput for Long Flows

## 3.7. Summary

From the evaluation results it is evident that the proposed scheme FA-DCTCP helps in significantly reducing the average FCT (up to 25.5%), 95th percentile of FCT (up to 29.7%)

and 99th percentile FCT (up to 32.5%) of short flows when compared to DCTCP. FA-DCTCP also ensures that the queue occupancies at the bottleneck links remain low. There is a slight negligible decrease (.04%) in throughput for long flows with this scheme; however, this trade-off is well worth the benefit achieved in terms of faster completion of latency sensitive short flows.

# Chapter 4 - Improving Throughput for Long Flows via SDN based ECN Adaptation in Data Centers

This chapter proposes a mechanism for improving throughput for long flows in data centers, by dynamically adapting ECN marking threshold in the in-network switches. It entails the data center to use any congestion control scheme that uses ECN (for example, DCTCP or ECN*) and achieve higher throughput for long flows. Key here is that ECN values are set dynamically as per the guidelines outlined in [44] to achieve better long flow throughput. As demonstrated by Alizedah et al. [11] and Wu et al. [44], there is a trade-off between long flow throughput and short flow latency. DCTCP mechanism achieves a balance between these by ensuring low latency for short flows while at the same time ensuring long flows achieve desired (though not optimal) throughput. The results presented in the DCTCP paper [11] shows there is still room for 10% to 15% improvement in long flow throughput, if long flows are to be considered in isolation and the marking threshold set with the objective to maximize long flow throughput. The same is also highlighted by Wu et al. in ECN tuning paper [44], which introduces a range of possible ECN marking thresholds with the corresponding trade-off between long flow throughput and short flow latency. Lower value for marking threshold helps to achieve lower latency for short flows but degrades long flow throughput. At the same time higher values for the marking threshold results in higher throughput for long flows but result in increasing the short flow latency. DCTCP uses a value for the ECN marking threshold which ensures low latency for short flows while the long flows achieve the desired throughput. The main point to note here is that there is still room for long flow throughput improvement by 10% to 15% if the marking threshold is set considering only long flows. The proposed mechanism outlines, a framework for dynamically adapting the ECN marking threshold such that: (a) It's set to a lower bound value in the presence of significant short flow to long flow mix, which guarantees low latency for short flows without degrading long flow throughput (ensuring long flow attains the desired throughput), and (b) It's set to a value that would aid in further improving long flow throughput when long flow dominates (i.e. when there are very few or no short flows at all) the link under consideration.

The proposed mechanism is based on ECN marking threshold tuning. ECN adaptation framework makes use of the Software Defined Networking (SDN) paradigm. SDN provides capabilities to observe the network and dynamically update network element configurations. The proposed mechanism leverages the knowledge about the network traffic mix garnered by the SDN controller, by virtue of being in the loop for every flow activation and termination, to determine the proportion of long and short flows in the data center network. Research studies [2], [18], [64] point out that traffic in data centers follow patterns related to Time-of-Day. One immediate use case of the proposed scheme would be, ECN adaptation leveraging the Time-of-Day traffic pattern, to achieve better throughput for long flows when long flows predominates the network and low latency for short flows in another scenario where short flows outnumber long flows. This could be achieved by, dynamically setting the predetermined ECN marking threshold values, based on the traffic pattern that would help to optimize the metric of interest for the observed traffic pattern.

Prototype based evaluations of the proposed mechanism show that, by dynamically adjusting ECN marking threshold associated with DCTCP, long flows are able achieve 12% improvement in throughput as opposed to the plain DCTCP mechanism that does not employ the proposed scheme. It also show that by adjusting ECN marking threshold associated with ECN* scheme, long flows are able to achieve 22% improvement in throughput. The impact is that data center network is able to accommodate 12% to 22% more long flows employing the proposed scheme.

In this chapter, proposed mechanism's design decisions (Section 4.1) are described. Its architecture (Section 4.2) and its operation (Section 4.3) are described in subsequent sections. Simulation and evaluation results are presented (Section 4.5) demonstrating the improvements the proposed scheme has to offer.

# 4.1. ECN Adaptation Framework Design Principles

The proposed mechanism is introduced as a system for dynamic adaptation of ECN, based on traffic flow patterns in a typical cloud data center. This adaptation framework is built using the Software Defined Networking paradigm. In this section, the proposed mechanism's design decisions are presented.

**Use Software Defined Networking Paradigm:** Proposed ECN adaptation framework is well suited for SDN based data centers for four reasons: (1) the SDN controller has a global view of the network (such as topology and routing information), (2) the SDN controller has the ability to monitor (observe) and collect relevant statistics (such as traffic matrix), (3) ECN adaptation framework is deployable as a straightforward SDN controller application, and (4) Availability of support for dynamically adjusting QoS parameters in the network nodes (for example Open Virtual Switches supporting dynamic QoS configuration via OpenFlow). Prototype implementation used in the thesis is based on OpenFlow.

**Leverage traffic patterns in Data center:** Studies [2], [18], [65]–[67] point out that traffic mix in data centers exhibits Time-of-Day related properties. Based on these observations data center operations over a day could be partitioned into different Time-of-Day windows. For example, assume that Time-of-Day is divided in to two windows to start with. One when there are a significant number of short flows (5h – 24h), and the second when there are very few short flows (24h – 5h). While inside the second window traffic pattern could be monitored and if it conforms to the expected pattern for this window then ECN threshold adaptation to higher values is done. As a safety mechanism such comparisons are done every 5 minute interval, and if it is observed that there are more number of short flows than expected, ECN threshold is switched back to the values that aid in achieving low latency for short flows.

**Ease of deployment:** A key consideration that went into the design of the solution was ease of deployability and adoption. This requires ensuring minimal changes to in-network elements. Taking advantage of the SDN controller's global view of the network and flow awareness, proposed solution could be deployed as a straightforward application in the SDN controller. Thus the solution does not require changes in the in-network elements especially the switches in the data center network. The approach proposed does not require any changes at all in switches or end hosts; can be deployed with any scheme that uses ECN to perform congestion control and could be deployed as an application on SDN controller.

## 4.2. ECN Adaptation Framework Architecture

Proposed mechanism gathers data about the underlying network state (especially network topology) and traffic flow composition (proportion of long flows to short flows). Towards this SDN controller's ability to observe the network state and collect traffic matrix is leveraged. ECN adaptation framework performs marking threshold adaptation decisions based on the aggregated information. Subsequently, ECN adaptation framework send triggers to the network elements (switches) to update the ECN marking threshold configuration for the bottleneck links that are impacted.

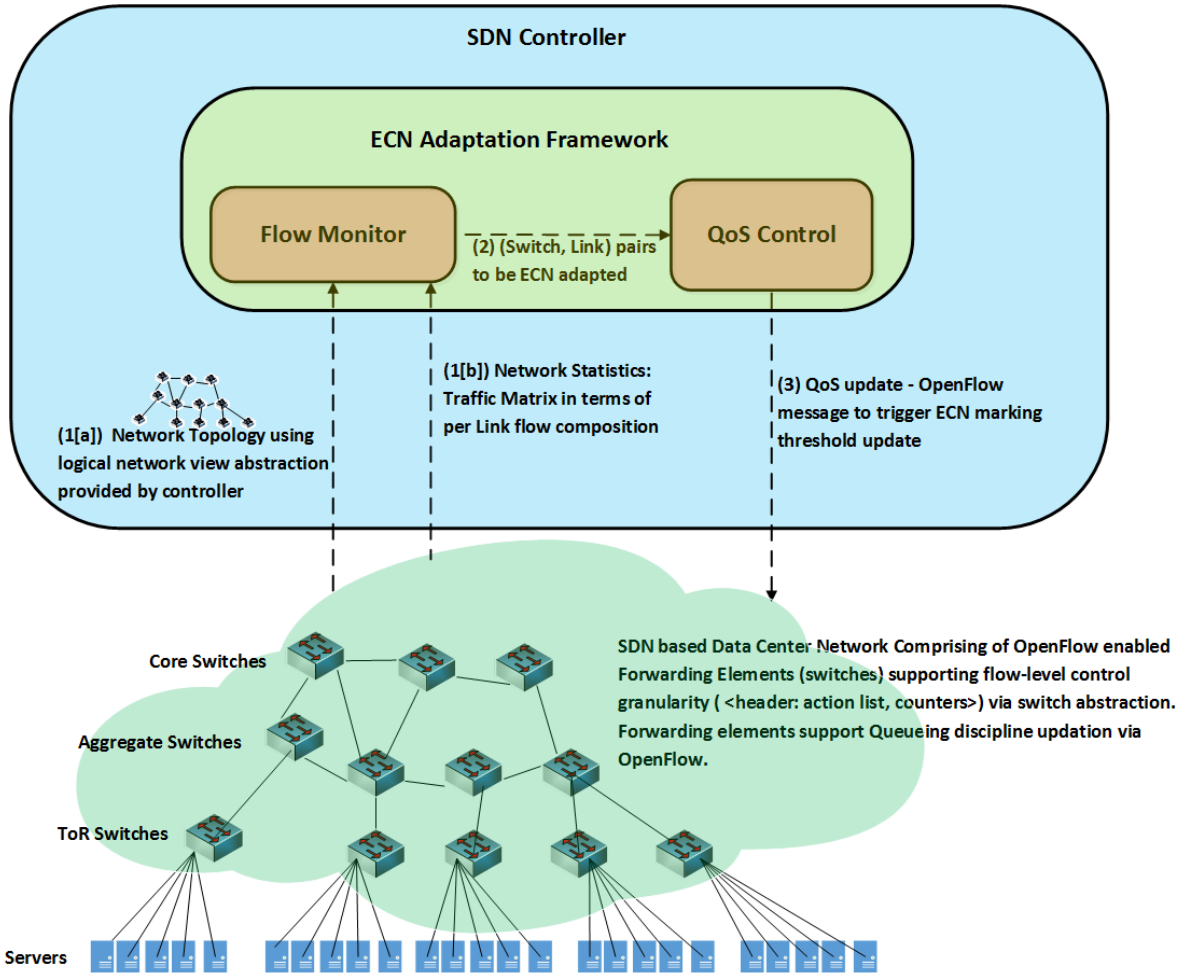


Figure 15 : ECN Adaptation Framework Architecture

Figure 15 presents the architecture of 'ECN adaptation framework'. As depicted, the proposed architecture has two main components - Flow Monitor Module and QoS Control Module. The framework is realized as a SDN application comprising of two modules:

(a) **Flow Monitor Module** performs the following functionalities.

1) Collects information required to make adaptation decisions, which comprises of underlying network topology and traffic matrix
2) Determines the bottleneck links where the adaptation has to be triggered considering the flow proportionality (traffic constituents)
3) Informs the QoS control modules about the list of (switch, link) pairs that are affected and the direction of adaptation (increase or decrease for the ECN marking threshold).

(b) **QoS Control Module** performs ECN threshold adaptation, by informing impacted switches about the marking threshold adaptation and the value to be used on adaptation.

## 4.3. ECN Adaptation Framework Operation

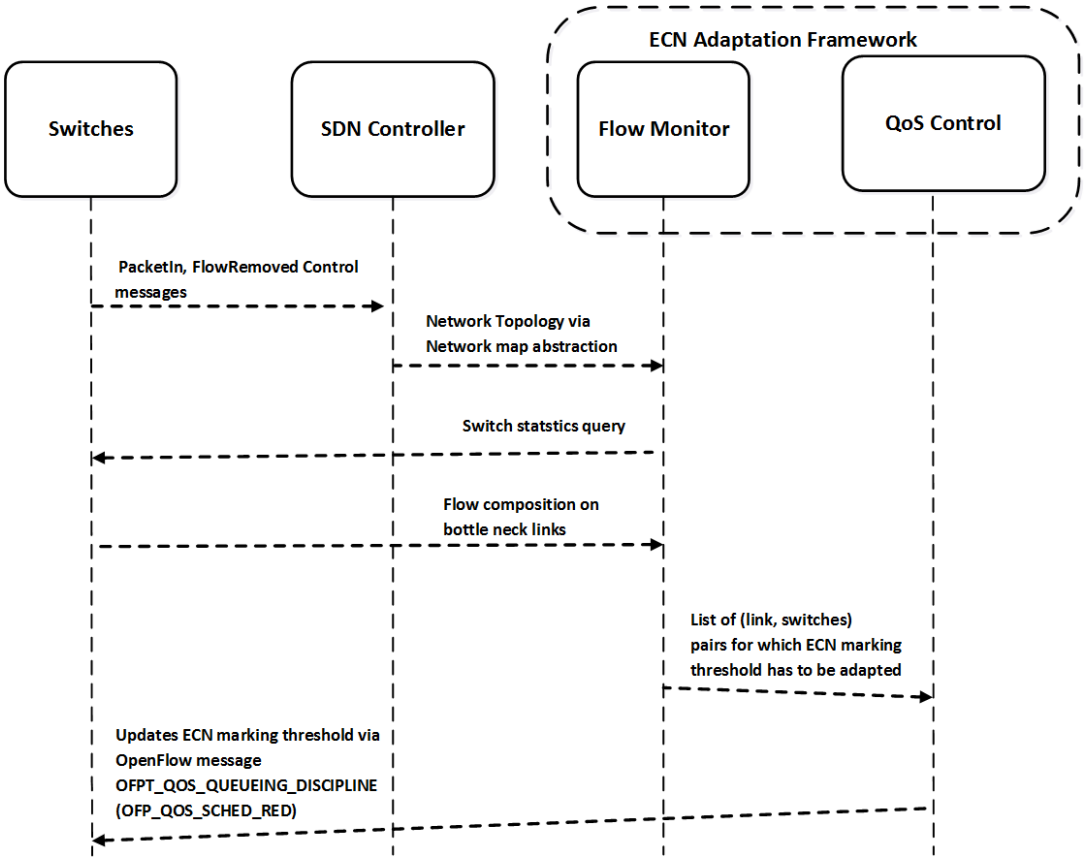


Figure 16 : ECN Adaptation Framework Operation

ECN adaptation framework periodically goes through a cycle of three steps: (i) data collection, (ii) traffic composition analysis and trigger generation, and (iii) ECN threshold adaptation. These steps are performed in a periodic fashion based on a pre-configured interval. Rest of the section describes the three steps in detail using Figure 16.

### 4.3.1. Data Collection

ECN adaptation framework relies on two types of data for its operation. First, it needs information about the network topology, especially the bottleneck links and the associated ToR switches in the topology. Since this information is readily available in the SDN controller, as part (subset) of the network map (topology) abstraction, the flow monitor module can easily access them. ToR switch links forms the bottleneck links in the topology. The list of bottleneck links could either be preconfigured or could be pulled dynamically from the network topology information available at the controller. Second, flow monitor module of the framework collects traffic statistics in-terms of flow composition ($\varphi(l)$ ratio of short flows to long flows) in the bottleneck links ($l$) and maintains flow composition table per bottleneck link per ToR switch. Flow monitor identifies the ratio of short flows to long flows ($\varphi(l)$) on a bottleneck link by periodically polling the flow counter statistics from the switches. For obtaining the flow statistics, two design alternatives are considered both of them taking advantage of SDN controller visibility in to the underlying network traffic patterns being at the central vantage point:

(1) **Periodic polling of switch flow tables:** Flow monitor periodically poll ToR switches to retrieve the flow counters. SDN switches maintain flow counters per port and allow such probes. For example the OpenFlow API provides mechanisms to poll and retrieve flow counters from OpenFlow switches [68]. This information combined with SDN controllers visibility into flow initiation, termination and the amount of bytes transferred over a flow is used by flow monitor to compute the short flow ratio to long flow ($\varphi(l)$).

(2) **Using five tuple to distinguish long flows at the SDN controller:** SDN controller is in the loop for all the flow initiations and terminations. This would allow SDN controller to

identify the type of flow based on the five tuple that uniquely identifies a long flow. Combining this with the flow routing information at the controller, Flow monitor derives the short flow to long flow ratio ($\varphi(l)$).

### 4.3.2. Trigger Generation

In this step flow monitor analyses the data collected in-order to make decision regarding the ECN adaptation triggers to be generated. The algorithm used by the flow monitor is captured in Figure 17. Traffic composition statistics is maintained by the flow monitor module as a list data structure with the following values in each node – switch ID, link ID, traffic composition as a ratio of short flow to long flows ($\varphi(l)$), current value of ECN threshold set on the link. Algorithm's key idea is to go through the list of switch, bottleneck link pairs checking the flow composition $\varphi(l)$ on them. If the ratio is below a preconfigured value denoted by 'delta' ($\delta$), then the ECN marking threshold for the bottleneck link is set to the upper bound value ($K_{UB}$), to allow long flows in them to attain better throughput giving them more aggressive or the ability to fill up the queues more. If the ratio of short flows is greater than $\delta$, marking threshold on bottleneck link is set to ECN marking threshold lower bound value ($\mathrm{K_{LB}}$), which offers short flows better latency but long flows to achieve their desired throughput.

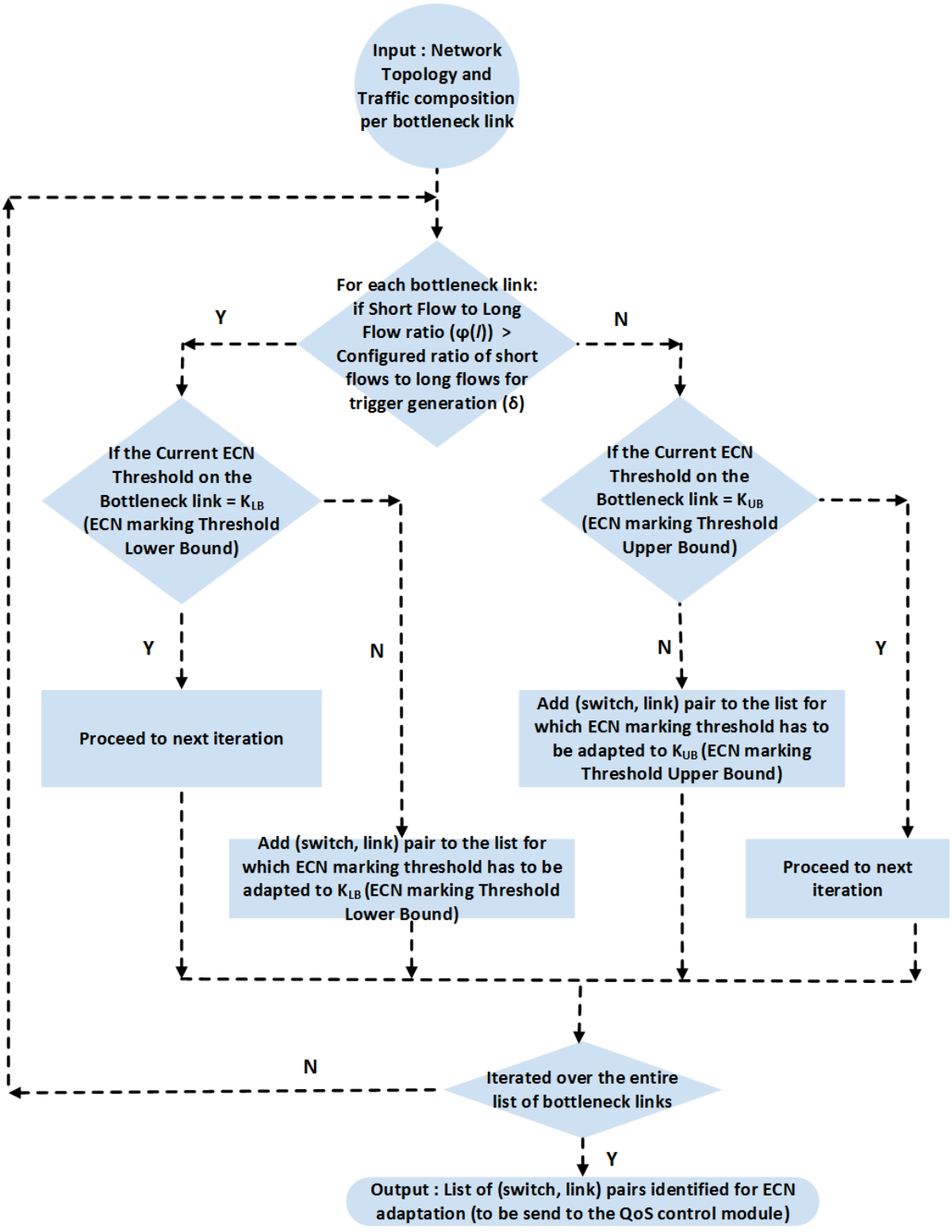


Figure 17: Flowchart of the logic used by the Flow Monitor component to make ECN adaptation trigger decisions

#### 4.3.2.1. Guidelines for Selecting Lower Bound and Upper Bound for ECN Marking Threshold

ECN adaptation framework is proposed for data centers employing ECN based congestion control mechanism. DCTCP and ECN* are two transport mechanism which employs such a scheme. The guiding principle or motivation behind the framework is that, setting a value for the ECN marking threshold to meet to competing goals (low latency for short flows and guaranteed throughput for long flows) often leads to trade-offs, where in long flow throughput tends to sacrifice about 10-15% of throughput when compared to a scheme targeted solely at maximizing throughput for long flows. Wu et al. [44] presents guidelines for identifying ECN marking threshold lower bound and upper bounds for ECN based marking schemes that use the instantaneous queue lengths to make marking decisions. Lower bound for the ECN threshold is identified as the marking threshold where the TCP performance is not affected after the congestion window reduction (for example because of buffer draining at the bottleneck link up on window reduction). The lower bound ($K_{LB}$) identified by authors in [44] corresponds to the well-known rule of thumb for drop tail buffer sizing presented in [69] and is given by the equation (3):

$$K_{LB} = (C \times T)/\sqrt{N} \qquad (3)$$

where $C$ is the bottleneck link capacity, $T$ is the average Round Trip Time for TCP connections in the network, and $N$ represents the number of TCP flows on the bottleneck link.

Alizadeh et al. [11] demonstrated that a data center network employing DCTCP as the congestion control mechanism could use a lower threshold ($K_{LB(DCTCP)}$) which is given by equation (4)

$$K_{LB(DCTCP)} = (C \times T)/7 \qquad (4)$$

Wu et al. [44] formulates the upper bound ($K_{UB}$) as given in equation (5) for switch with $B$ as the buffer size apportioned per port.

$$K_{UB} = \frac{1}{2}(B - C \times T) \qquad (5)$$

The minimum value of switch buffer size (B), required to avoid buffer over flow is given in equation (6):

$$B = 3C \times T \qquad (6)$$

Gist of the idea here is to set ECN marking threshold at the bottleneck link to the lower bound in the presence of a traffic pattern where there is a significant mix of short flows to long flows. In scenarios where the traffic on the bottleneck link comprises mainly of long flows and very few short flows ECN marking threshold set to the upper bound value with the aim to maximize long flow throughput.

#### 4.3.2.2. Guidelines for Selecting Short Flows to Long Flow Ratio for Trigger Generation

Flow monitor bases its trigger for ECN adaptation on the traffic flow composition, expressed in terms of short flows to long flows ratio (δ), on the bottleneck link under consideration for the monitoring period (interval). Choice of δ plays a very important role in ensuring the fine balance between the requirements of long flows and short flows are met. Based on the studies presented in papers [11], [18], [44], [70], δ is set as 1 corresponding to very few short flows in the link, bounded by a maximum of 4 long flows that could co-exist in a bottleneck link in typical cloud data center.

When ratio of short flows to long flows ($\varphi(l)$) is less than or equal to δ on a bottleneck link, flow monitor sends a trigger to set the ECN marking threshold on the link to the upper bound ($K_{UB}$).

$$i.e.\ if\ \big(\varphi(l)\big) \leq\ \delta;\ set\ ECN\ threshold\ trigger\ and\ marking\ threshold\ to\ K_{UB}$$

When ratio of short flows to long flows ($\varphi(l)$) is greater than δ on a bottleneck link, flow monitor sends a trigger to set the ECN marking threshold on the link to the lower bound ($K_{LB}$)

$$i.e.\ if\ \big(\varphi(l)\big) > \delta;\ set\ ECN\ threshold\ trigger\ and\ ECN\ marking\ threshold\ to\ K_{LB}$$

### 4.3.3. ECN Threshold Adaptation

Final step in ECN adaptation framework operation is to change the ECN marking threshold on the impacted bottleneck links, based on the triggers generated by the flow monitor module. Flow monitor hands over the list of switch, link pairs that are to be ECN adapted along with the corresponding ECN threshold values to be set to QoS Control Module. QoS control module in-turn conveys the ECN adaptation decision to the switches. Towards this QoS Control module sends out QoS re-configuration messages, via OpenFlow, to switches asking them to update the ECN marking threshold on the bottleneck links that are impacted. Since ECN is implemented based on the RED queuing discipline (scheduling), this step would require SDN switches that support configuration of the QoS (queueing discipline) parameters via the SDN controller. One such SDN switch architecture is QoSFlow [71], which extends OpenFlow datapath part of OpenFlow switches, adding capabilities for manipulating queuing disciplines via custom OpenFlow QoS configuration messages (of type OFPT_QOS_QUEUEING_DISCIPLINE) from the SDN controller. Another approach would be to extend the Open Virtual Switch's queuing disciplines to support RED queuing discipline and control its configuration via either OVSDB or OpenFlow. For the prototype based implementation and evaluation, this work uses the OpenFlow QoS framework provided by QoSFlow. SDN switches receives the QoS configuration message and updates the ECN settings for the bottleneck link using the threshold specified, setting both the low and high threshold associated with ECN on the switches to the specified value thereby ensuring they mark packets based on instantaneous queue occupancy.

### 4.3.4. Choosing the Time Period for ECN Adaptation Framework Operation

Nature of data center Traffic plays a major role in determining the time period for deploying ECN adaptation operation. One possibility is to have the framework execute its operation in specific Time-of-Day, when the short flows to long flows ratios are observed to follow specific patterns. The other approach is to have the framework running throughout the day and make adaptation decisions at a time scale of 10 minutes, which is the short term time scale for traffic predictability in data center networks reported in [2], [18], [65].

As for the first approach, it could be assumed that data center traffic workloads follow the sun as depicted in papers [2]. This means that there are periods when the number of short flows in a data center is very less and the long flows predate the network especially the bottleneck links. For example a case similar to the one mentioned in related research [2], [18], [64]–[66], which indicates the presence of data center traffic patterns that depends on the Time-of-Day, could be considered. One such possibility where the operation time lines of ECN adaptation framework is split into two main windows, one from 5h-24h where in the number of short flows are significantly higher than long flows (it could be anywhere between 2:1 to 20:1). The second window where there is equal (or lesser) number of short flows and long flows (representing a 1:1 proportionality) normally observed during the small hours of the night, 24h-5h, when majority of the operation involve VM backup, data structure synchronization, VM migration etc. This window opens up a time period where the framework could be put into action as a starting point.

The second approach would be to perform ECN adaptation in a more find grained time scale granularity. The best bet in this case would be time windows of 10 minutes, during which data center network traffic is shown to be stable and predictable as per studies [2], [18], [65]. The framework since it is built based on SDN offers considerable flexibility in scaling this solution for fine grained windows of 10 minutes.

## 4.4. Evaluation and Analysis

### 4.4.1. Evaluation Criteria

The evaluation of ECN adaptation framework is done via tests conducted in a topology realized using Mininet environment. The objective of tests is to identify the gain in throughput achieved by longs flows by adopting the proposed scheme. Tests are conducted for DCTCP, ECN*, DCTCP (with ECN adaptation) and ECN* (with ECN adaptation). Tests are conducted to cover scenarios with different number of long flows and for different long flow sizes to quantify their impact.

Three different scenarios are tested with the objective to identify the improvements that could be achieved via deploying ECN adaptation scheme. First test scenario covers DCTCP

and ECN* schemes with and without ECN adaptation. Second test scenario analyses the impact of the number of long flows on the bottleneck link. Third scenario covers the impact of long flow sizes on the performance of the scheme.

### 4.4.2. Test Topology, Test Setup and Related Parameters

The testbed depicted in Figure 18 consists of a Top-of-Rack switch connected to 5 servers. The reason for limiting this to 5 servers is the observation from [11], [44] that the maximum number of concurrent long flows on a bottleneck link will be 4. Servers S1 through S4 acts as the senders whereas S5 acts the receiver. Test parameters for the three test scenarios are captured in Table 2 and Table 3.

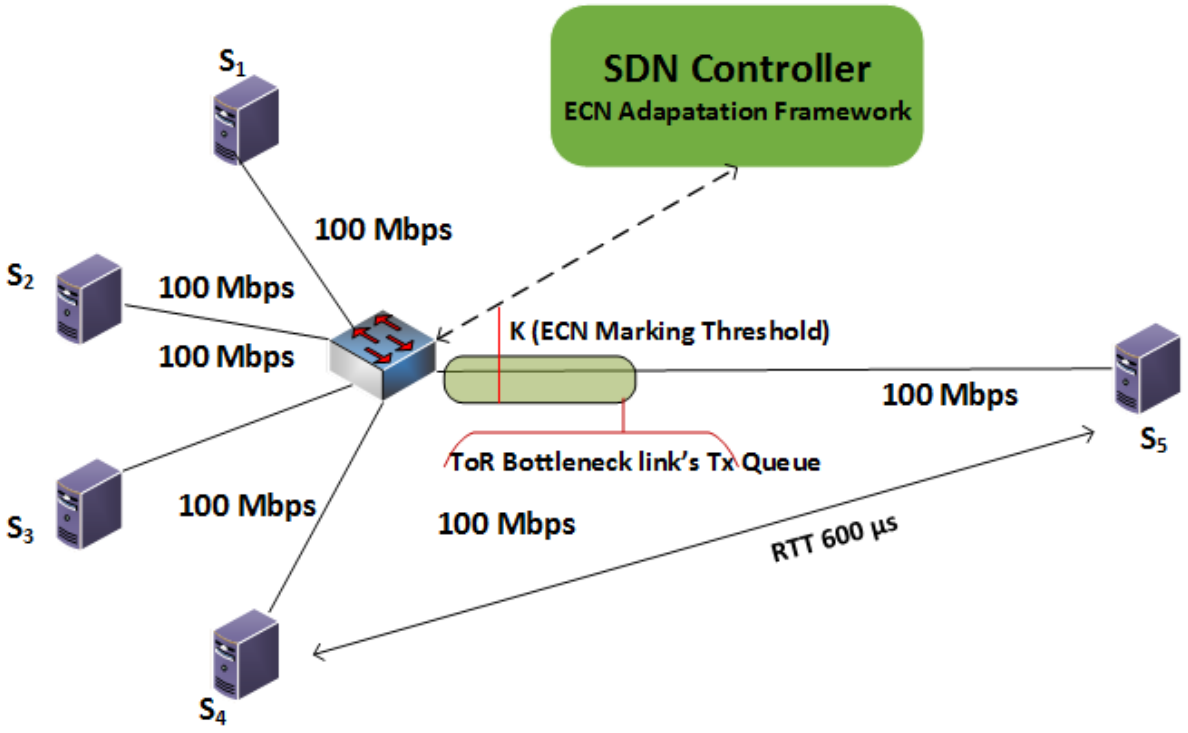


Figure 18 : Test setup for ECN adaptation framework

Tests were performed using Mininet running on a Ubuntu LTS box with the following configuration - Intel Core i5 with two 2.9 GHz cores and 16 GB of RAM. As alluded to in [61], Mininet is well suited for testing network limited scenarios especially congestion control and adaptation scenarios. Mininet is the mostly widely used tool for testing SDN and OpenFlow based proposals and solutions.

Table 2 : Test setup parameter settings for DCTCP based tests

| Parameters | Value |
|---|---|
| Link speed | 100Mbps |
| Link Latency | 150µs |
| Round Trip Time | 600µs |
| Switch Buffer Size | 25 Packets |
| ECN Marking Threshold Lower Bound | 3 Packets |
| ECN Marking Threshold Upper Bound | 10 Packets |
| Exponential averaging factor | 1/16 |

For DCTCP based experiments the lower and upper bounds of ECN marking thresholds are calculated using equations (4) and (5) respectively. For ECN* based experiments the lower and upper bounds of ECN marking thresholds are calculated using equations (3) and (5) respectively. The minimum switch buffer size required per port is calculated using equation (6).

Table 3: Test setup parameter settings for ECN* based tests

| Parameters | Value |
|---|---|
| Link speed | 100Mbps |
| Link Latency | 150µs |
| Round Trip Time | 600µs |
| Switch Buffer Size | 25 Packets |
| ECN Marking Threshold Lower Bound | 5 Packets |
| ECN Marking Threshold Upper Bound | 10 Packets |

## 4.4.3. Test Results and Discussion

This section presents the results from the six test scenarios. Each of the tests was run 50 times and the averages of results from these test runs are captured.

### 4.4.3.1. DCTCP with ECN adaptation

To evaluate and compare the performance of DCTCP to DCTCP with ECN adaptation, two senders originate two long flows (using iperf traffic generator) transferring 1000MB each. Parameters used for the test are captured in Table 2.

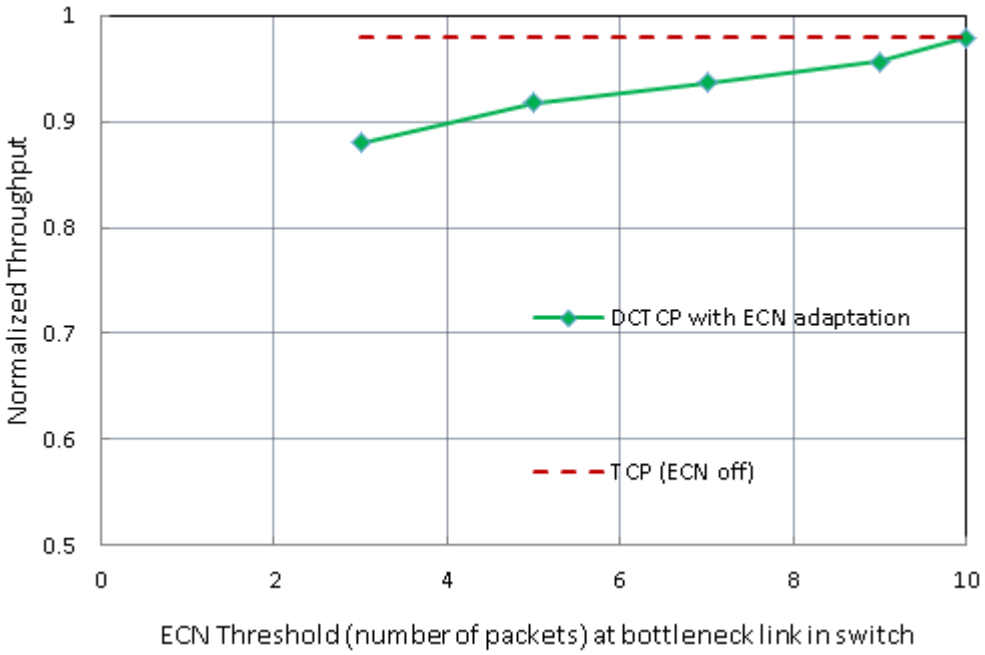


Figure 19: DCTCP with ECN adaptation

Figure 19 above depicts results from the tests and show DCTCP with ECN adaptation achieves a 12% improvement over DCTCP in terms of long flow throughput. This gain in performance was achieved, by setting the marking threshold to a value of 10 packets which is the upper bound for ECN marking [deduced from equation (4)], as opposed to the lower bound of 3 packets that favours low latency operation. This scheme effectively helps in accommodating up to 12% more long flows in comparison to a scheme that does not use ECN adaptation.

### 4.4.3.2. ECN* with ECN adaptation

To evaluate and compare the performance of ECN* to ECN* with ECN adaptation, two senders originate two long flows (using Iperf traffic generator) transferring 1000MB. Parameters used for the test are captured in Table 3.

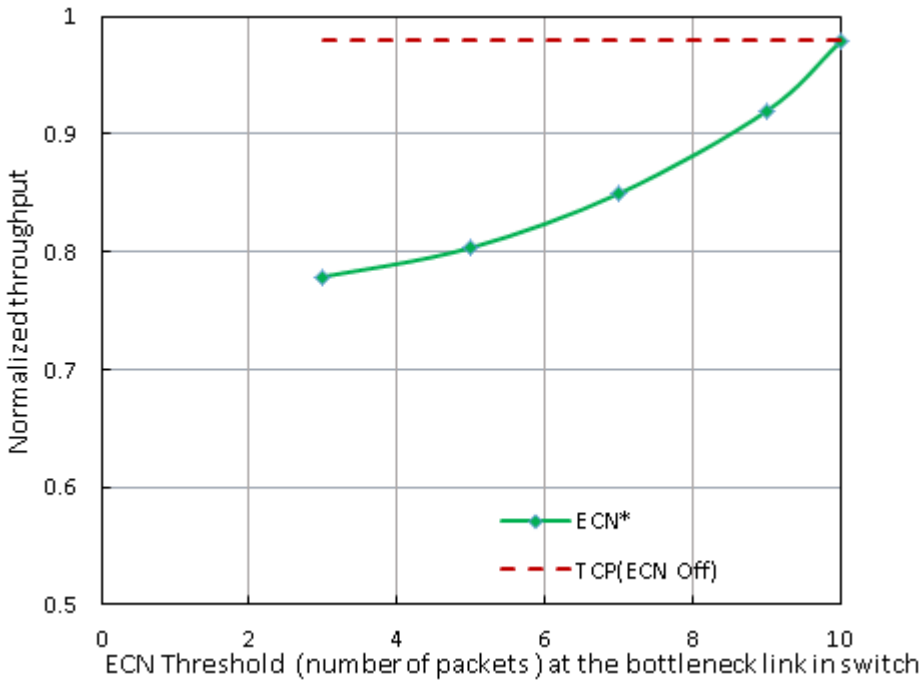


Figure 20: ECN* with ECN adapatation

Figure 20 depicts results from the tests that show ECN* employing ECN adaptation scheme achieves a 22% improvement over plain ECN*. This gain in performance was achieved by setting the marking threshold to a value of 10 packets which is the upper bound of ECN marking (deduced from equation (4)) as opposed to the lower bound value of 5 packets. This suggests the ability of this scheme to accommodate higher number of long flows in comparison to plain ECN*.

#### 4.4.3.3. DCTCP with ECN Adaptation: Impact of Number of Long Flows on the Bottleneck Link

To understand the impact of multiple long flows traversing the bottleneck link on DCTCP with ECN adaptation, tests were performed by varying the number of senders from 2 to 4. Each of the servers participating in the test starts a long flow (using iperf traffic generator) to the receiver R transferring 1000MB over the duration of the test run. Parameters used for the test are captured in Table 2.

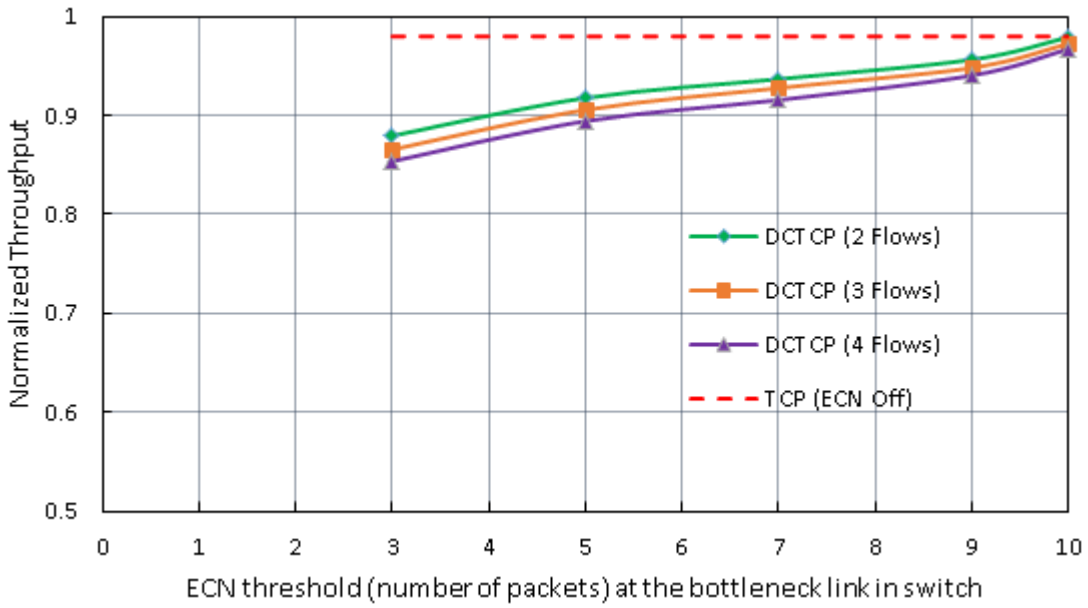


Figure 21 : DCTCP with ECN adaptation when multiple long flows traverse bottleneck link concurrently

Figure 21 shows that shows that there is a slight degradation in throughput when the number of long flows concurrently traversing a bottleneck link is increased; this is in line with the behaviour mentioned in [11]. The key take away from this test is that, ECN adaptation scheme accommodates the expected range of long flows that could concurrently traverse a bottleneck link. It also helps in achieving up to 12% improvement in long flow throughput, by virtue of dynamic adjustment of marking threshold to the upper bound value of 10 (packets) as opposed to the value 3 (packets) that is used by plain DCTCP.

#### 4.4.3.4. ECN* with ECN Adaptation: Impact of Number of Long Flows on the Bottleneck Link

To understand the impact of multiple long flows traversing the bottleneck link on ECN* with ECN adaptation; test are conducted with varying number of senders from 2 to 4. Each of the servers participating in the test starts a long flow (using iperf traffic generator) to the receiver R transferring 1000MB over the duration of the test run. Parameters used for the test are captured in Table 3.

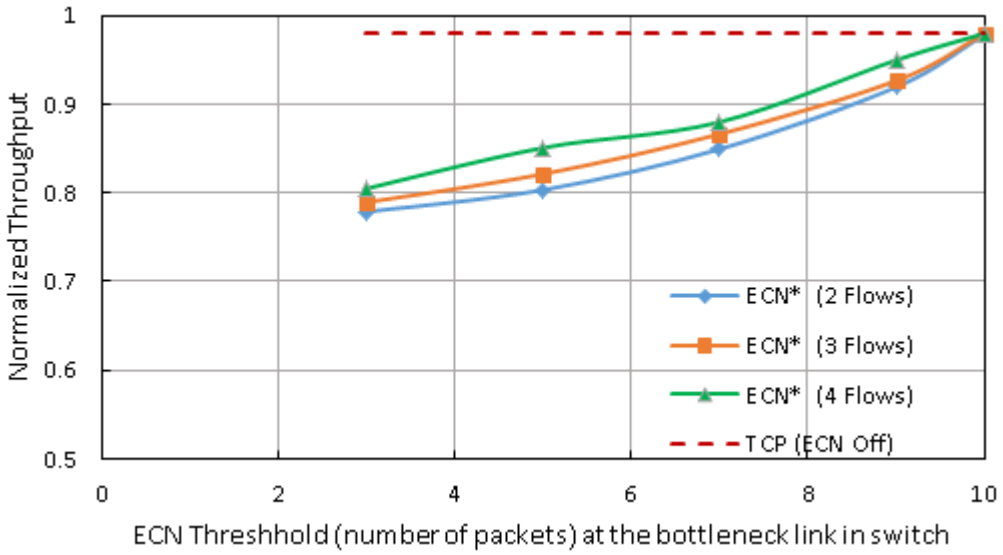


Figure 22 : ECN* scheme with ECN adaptation when multiple long flows traverse bottleneck link concurrently

Figure 22 depicts results from the tests that show that there is a slight improvement in throughput when the number of long flows traversing the bottleneck link increases; this is in line with the behavior mentioned [44]. Results from the test implies that use of ECN adaptation scheme is beneficial, taking into consideration the possible combination of the number of long flows that could traverse a bottleneck link. The test results also helps to ascertain the fact that ECN adaptation scheme helps achieve up to 22% improvement in the case of ECN* scheme, by virtue of dynamic adjustment of marking threshold to the upper bound value of 10 packets from the lower bound value of 5 packets that is employed by plain ECN* scheme.

### 4.4.3.5. DCTCP with ECN adaptation: Impact of Long Flow Sizes

To analyze the impact of size of long flows on DCTCP with ECN adaptation; tests were conducted with number of senders sets as 2 and by varying the size of the data send by them to 10MB, 100MB, 500MB and 1000MB. Parameters used for the test are captured in Table 2.

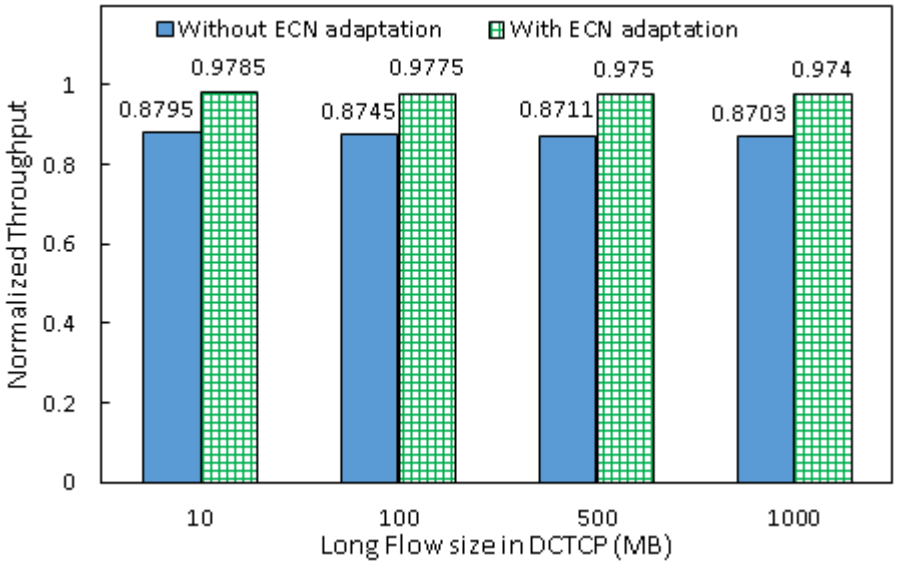


Figure 23 : Impact of long flow sizes on DCTCP with ECN adaptation

Figure 23 depicts results from the tests that show there is a negligible change in throughput when the size of the long flow varies. This result implies that that long flow size would not be impacting the performance of the proposed ECN adaptation scheme and the scheme could be applied effectively to the entire range of long flow sizes, ranging from 10MB to 1GB. The test results also helps in ascertaining that, usage of ECN adaptation helps to improve long flow throughput by up to 12% in the case of DCTCP.

### 4.4.3.6. ECN* with ECN adaptation: Impact of Long Flow Sizes

To analyze the impact of size of long flows on ECN* with ECN adaptation, tests were conducted with number of senders sets as 2 and by varying the size of the data send by them as follows 10MB, 100MB, 500MB and 1000MB. Parameters used for the test are captured in Table 3.

Figure 24 shows that there is a negligible change in throughput when the size of the long flow varies. This result implies that that long flow size would not be impacting the performance of the proposed ECN adaptation scheme in the case of ECN* and the scheme could be applied effectively to the entire range of long flow sizes, ranging from 10MB to 1GB. The test results also helps in ascertaining that, usage of ECN adaptation helps to improve long flow throughput by up to 22% in the case of ECN*.

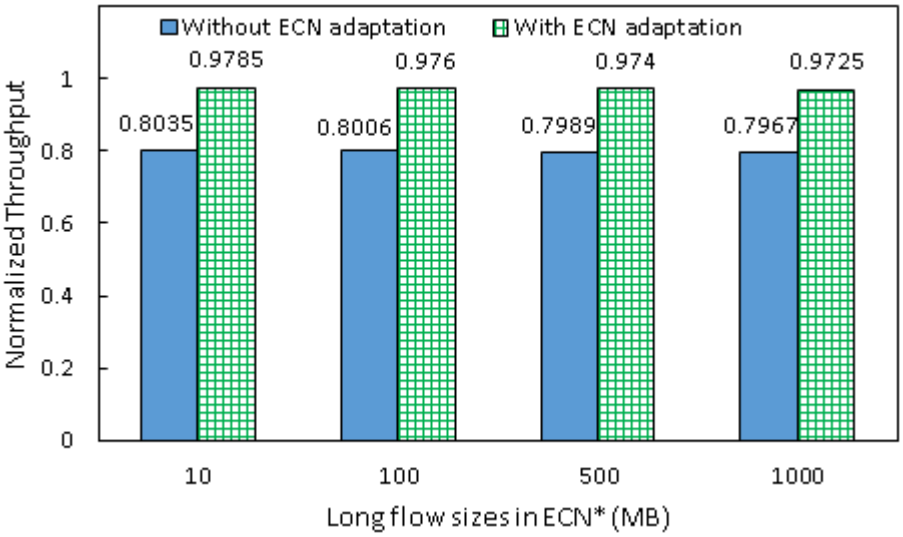


Figure 24 : Impact of long flow sizes on ECN* with ECN adaptation

## 4.5. Summary

ECN adaptation framework leverages the capabilities of the SDN paradigm – ability to observe and dynamically update the network based on traffic conditions. It demonstrates that the traffic patterns of the data center networks provide room for tuning network element configuration, to suit the predicted deterministic patterns. The proposed framework helps to improve performance expected out of the network, by applying configuration values (devised based on guidelines to suit the instantaneous network state) on the fly to network elements. ECN adaptation framework deployed in a data center network running DCTCP has the potential to improve long flow throughput by up to 12% and thereby could accommodate up to 12% more long flows. In the case of data centers employing ECN* scheme this would go up to 22% with the proposed framework.

# Chapter 5 - Conclusions and Future Work

This chapter summarises the thesis work by explaining contributions of the thesis research and outlines related future research directions.

## 5.1. Summary of Work

This thesis presented a synopsis of recent data center transport protocols that that have been proposed to improve the data center network performance – either by achieving low latency, high throughput, reducing deadline miss or a combination of the three. In addition, thesis also presented two new schemes to improve data center transport performance. Both these schemes require minimal changes, and are therefore deployment friendly, in a single administrative data center environment.

Chapter 3 of this thesis, proposed FA-DCTCP [14], a scheme designed ensuring that it is easier to implement and deploy with minimal changes at the end-hosts and without requiring any changes to the network fabric. FA-DCTCP requires only 52 lines of code change to congestion control algorithm at the DCTCP sender and do not require any change to the DCTCP receiver or ToR switch. Another point to note is that FA-DCTCP, similar to DCTCP, is tailored for the data center environment and deals with traffic internal to the data center. Characteristics of data center environment differ significantly from the wide area networks. Especially the network will be under single administrative control and will be mostly homogeneous. FA-DCTCP leverages this aspect to its advantage in identifying the flow types and also for ensuring deployment friendliness. FA-DCTCP evaluations show that, a less proportionate reduction of congestion window corresponding to the connection associated to short flow, in comparison to the congestion window associated with long flow, in the presence of congestion could ensure better flow completion time for short flows while not degrading the throughput of long flows. Emulations and tests conducted using data center workloads show an improvement of up to 32.5% in $99^{th}$ percentile of flow completion time and thereby up to 32.5% reduction in $99^{th}$ percentile of latency for short flows. Tests also reveal that improvement in FCT for short flows are without incurring any additional increase

in the queue occupancy at the bottleneck links and at the expense of a very minor decrease .04% in the throughput of long flows.

Chapter 4 of the thesis proposed a SDN based dynamic ECN adaptation mechanism for improving throughput for long flows in data centers. It does not require changes in end-hosts and network elements, and could be deployed as a simple application in the SDN controller. The scheme by virtue of not being tied to any particular congestion control mechanism entails flexibility and ease of adoption. Mininet based evaluation results show that long flows are able achieve 12% improvement in throughput as opposed to the plain DCTCP mechanism that does not employ the proposed scheme. In a ECN* setting, long flows are able to achieve 22% improvement in throughput. Thus this scheme makes it possible for the data center network to accommodate 12% to 22% more long flows.

## 5.2. Conclusions

This thesis research demonstrate that, by incorporating flow awareness in to data center transport protocol it is possible to achieve short flow prioritization and thereby significant reduction in latency observed by short flows in data center networks. This thesis research also show that, by leveraging Software Defined Networking paradigm and it's capabilities to observe network traffic state and dynamically adapt QoS parameters of network elements, long flows in data center networks could achieve better throughput.

## 5.3. Future Work

The schemes proposed in this work are based on single path data center transport protocols. Recent research proposals like MPTCP and XMP, have explored the possibility of leveraging the availability of multiple paths between sender and receiver, typical in data center topologies to improve performance. Analysis on whether flow awareness could be used to improve these protocols, especially MPTCP which is reported to have significant degradation in performance while handling short flows of size less than 70KB, looks to be worth exploring.

ECN adaptation framework proposed in chapter 4 is designed assuming data center network employing a single SDN controller. There are number of recent proposals for

distributed SDN controller architectures, ONOS[72] and OpenDayLight [73], which are designed with the aim of improving availability and scalability of SDN controller and the solutions deployed on top of it. Scalability of the proposed ECN adaptation framework can be further improved by extending it to work in a setting with distributed controllers.

# References


[1] D. Abts and J. Kim, "High performance datacenter networks: Architectures, algorithms, and opportunities," in *Synthesis Lectures on Computer Architecture*, 2011, pp. 1–115.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 267–280.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009, pp. 51–62.

[4] 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*, 2009, pp. 202–208.

[5] "Marissa Mayer at Web 2.0." [Online]. Available: http://glinden.blogspot.ca/2006/11/marissa-mayer-at-web-20.html. [Accessed: 22-Sep-2014].

[6] "Amazon found every 100ms of latency cost them 1% in sales." [Online]. Available: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/. [Accessed: 22-Sep-2014].

[7] "Velocity and the Bottom Line." [Online]. Available: http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html. [Accessed: 22-Sep-2014].

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, 2008, pp. 63–74.

[9] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009, pp. 63–74.

[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*, 2009, pp. 73–82.

[11] 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*, 2010, pp. 63–74.

[12] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 127–138.

[13] K. Ramakrishnan, S. Floyd, and D. Black, "RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP."

[14] S. Joy and A. Nayak, "Improving Flow Completion Time for Short Flows in Data Center Networks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, 2015.

[15] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009, pp. 39–50.

[16] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit Multipath Congestion Control for Data Center Networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 73–84.

[17] Y. Zhang and N. Ansari, "On Architecture Design, Congestion Notification, TCP Incast and Power Consumption in Data Centers," *Commun. Surv. Tutorials, IEEE*, vol. 15, no. 1, pp. 39–64, 2013.

[18] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, 2011, pp. 8:1–8:12.

[19] D. Abts and B. Felderman, "A Guided Tour of Data-center Networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

[20] S. Brin and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine," in *Proceedings of the Seventh International Conference on World Wide Web 7*, 1998, pp. 107–117.

[21] 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," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, 2007, pp. 205–220.

[22] R. Cheveresan, M. Ramsay, C. Feucht, and I. Sharapov, “Characteristics of Workloads Used in High Performance and Technical Computing,” in *Proceedings of the 21st Annual International Conference on Supercomputing*, 2007, pp. 73–82.

[23] “SGI Open Source XFS. XFS: A High-performance Journaling Filesystem.” [Online]. Available: http://oss.sgi.com/projects/xfs/. [Accessed: 29-Nov-2014].

[24] “Apache Hadoop Project.” [Online]. Available: http://hadoop.apache.org/. [Accessed: 29-Sep-2014].

[25] 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*, 2007, pp. 59–72.

[26] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004, p. 137–149.

[27] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, 2010, p. 19.

[28] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, “Myrinet: a gigabit-per-second local area network,” *Micro, IEEE*, vol. 15, no. 1, pp. 29–36, Feb. 1995.

[29] “InfiniBand Architecture Volume 2, Release 1.3.” [Online]. Available: http://www.infinibandta.org/content/pages.php?pg=technology_public_specification. [Accessed: 15-Nov-2014].

[30] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “Towards a Next Generation Data Center Architecture: Scalability and Commoditization,” in *PRESTO Workshop at SIGCOMM*, 2008.

[31] “Cisco Data Center Infrastructure 2.5 Design Guide.” [Online]. Available: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.pdf. [Accessed: 27-Oct-2014].

[32] “Cisco Global Cloud Index : Forecast and Methodology , 2013 – 2018,” *Cisco White Paper*, 2013. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html. [Accessed: 12-Dec-2014].

[33] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized Task-aware Scheduling for Data Center Networks,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 431–442.

[34] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009, pp. 303–314.

[35] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding Long Tails in the Cloud," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013, pp. 329–342.

[36] A. Bouch, A. Kuchinsky, and N. Bhatti, "Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2000, pp. 297–304.

[37] D. A. Patterson, "Latency Lags Bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, Oct. 2004.

[38] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, "It's Time for Low Latency," in *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, 2011, p. 11–15.

[39] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012, p. 253–266.

[40] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 139–150.

[41] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "DIBS: Just-in-time Congestion Mitigation for Data Centers," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 6:1–6:14.

[42] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, 2014, pp. 149–160.

[43] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, Not Foes: Synthesizing Existing Transport Strategies for Data Center Networks," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 491–502.

[44] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for Data Center Networks," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, pp. 25–36.

[45] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 2157–2165.

[46] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 115–126.

[47] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never Than Late: Meeting Deadlines in Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 50–61.

[48] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-optimal Datacenter Transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, pp. 435–446.

[49] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 266–277.

[50] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012, p. 399–412.

[51] Open Networking Foundation, "SDN Architecture Overview," 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf. [Accessed: 02-Sep-2014].

[52] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," *White Pap. Palo Alto, US Open Netw.*, 2012.

[53] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter.," in *HotNets*, 2009.

[54] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live Migration of an Entire Network (and Its Hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 109–114.

[55] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking End-to-end Congestion Control in Software-defined Networks," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 61–66.

[56] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 1629–1637.

[57] H. Wu, Z. Feng, C. Guo, and Y. Zhang, “ICTCP: Incast Congestion Control for TCP in Data-Center Networks,” *Networking, IEEE/ACM Trans.*, vol. 21, no. 2, pp. 345–358, Apr. 2013.

[58] M. Casado and J. Pettit, “Of Mice and Elephants,” 2013. [Online]. Available: http://networkheresy.com/2013/11/01/of-mice-and-elephants/. [Accessed: 28-Aug-2014].

[59] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software Defined Wan,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, pp. 3–14.

[60] “Datacenter TCP.” [Online]. Available: http://simula.stanford.edu/~alizade/Site/DCTCP.html. [Accessed: 28-Sep-2014].

[61] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, “Reproducible Network Experiments Using Container-based Emulation,” in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, pp. 253–264.

[62] B. Lantz, B. Heller, and N. McKeown, “A Network in a Laptop: Rapid Prototyping for Software-defined Networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 19:1–19:6.

[63] “Iperf - The TCP/UDP Bandwidth Measurement Tool.” [Online]. Available: https://iperf.fr/. [Accessed: 28-Sep-2014].

[64] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, “ElasticTree: Saving Energy in Data Center Networks,” in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, 2010, p. 17.

[65] T. Benson, A. Anand, A. Akella, and M. Zhang, “The Case for Fine-grained Traffic Engineering in Data Centers,” in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, 2010, p. 2.

[66] K. K. Nguyen, M. Cheriet, and Y. Lemieux, “Virtual Slice Assignment in Large-Scale Cloud Interconnects,” *IEEE Internet Comput.*, vol. 18, pp. 37–46, 2014.

[67] K.-K. Nguyen, M. Cheriet, M. Lemay, M. Savoie, and B. Ho, "Powering a Data Center Network via Renewable Energy: A Green Testbed," *Internet Comput. IEEE*, vol. 17, no. 1, pp. 40–49, Jan. 2013.

[68] Open Networking Foundation Open Flow Consortium, "OpenFlow Switch Specification Version 1.4.0," 2013.

[69] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2004, pp. 281–292.

[70] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 151–162.

[71] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, 2013, pp. 81–86.

[72] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 1–6.

[73] "Open Daylight Project." [Online]. Available: http://www.opendaylight.org/. [Accessed: 01-Dec-2014].