Transport Protocols for Data Center Communication

Evisa Tsolakou

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo

Thesis supervisor:

Prof. Jörg Ott

Thesis advisor:

Lect. Pasi Sarolahti

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Evisa Tsolakou

Title: Transport Protocols for Data Center Communication

Date: Language: English Number of pages: 10+59

Department of Communications Engineering

Professorship: Networking Technology

Supervisor: Prof. Jörg Ott

Advisor: Lect. Pasi Sarolahti

Data centers are becoming more and more important since there is a number of services covered especially by them. At the same time it is reasonable to maintain the costs of data centers low from a number of perspectives.
To this end, one could propose a number of changes in the data center environment. While there is a number of studies that focus on different aspects of the data center environment, one of the most important factors that can be studied and changed is the transport protocol used in the data center environment. This change will have an impact on a number of factors in the data centers.
For the purpose of this thesis a number of transport protocols were studied, starting from the broadly used TCP to a number of especially designed for data centers ones. These variations were studied for the changes they impose and the positive results they bring.
At the same time the significance of DCTCP, the most extensively studied and deployed data center environment protocol was made apparent and the positive results from its deployment. This study outlines the necessity to know its behaviour while coexisting with TCP as well since its deployment in the wide Internet would bring positive results for latency, losses and buffer queues minimization.
To this end, the protocol was studied by emulating network behaviour in Mininet network emulator and it was found out that its coexistence with TCP is possible without the TCP traffic starving as long as some parameters settings are followed.

Keywords: Data Center Networks, Data Center Traffic, Transport Protocols, Congestion Control, Queue Management, DCTCP

Preface

I would like to express my sincere gratitude to my thesis advisor Pasi Sarolahti for taking on the role of my advisor and helping me step by step during this thesis with his valuable comments, continuous guidance, deep knowledge and especially patience during the process of writing.

I would like to thank in addition my thesis professor Jörg Ott for allowing me to write the topic and for his patience during the process of writing as well.

I would like to thank in addition the professors and staff of the Department of Communications Engineering-Networking Technology for all these years of support and study.

Last, I would like to express my love to my family and my friends for constant support, good times and holding my hand through all the times that meant something in my life. Being in different parts of the world does not keep us apart.

Contents

Abbreviations

NIST	National Institute of Standards and Technology
ToR	Top of Rack
NIC	Network Interface Card
SNMP	Simple Network Management Protocol
TCP/IP	Transmissions Control Protocol/Internet Protocol
OSI	Open Systems Interconnection
SEQ	Sequence Number
URG	Urgent
ACK	Acknowledgement
PSH	Push
RST	Reset
SYN	Synchronize
FIN	Finish
CWND	Congestion Window
AIMD	Additive Increase Multiplicative Decrease
RTT	Round Trip Time
EWMA	Exponential Weighted Moving Average (EWMA)
CA	Congestion Avoidance
RTO	Retransmission Timeout
ICTCP	Incast congestion Control TCP
MPTCP	Multipath TCP
RCP	Rate Control Protocol
D^3	Deadline-Driven Delivery
PDQ	Preemptive Distributed Quick
PASE	Prioritization, Arbitration and Self-adjusting Endpoints
AQM	Active Queue Management
RED	Random Early Detection
CoDel	Controlled Delay
PIE	Proportional Integral controller Enhanced
ECN	Explicit Congestion Notification
CE	Congestion Experienced
ECT	ECN-Capable Transport
ECE	ECN-Echo
CWR	Congestion Window Reduced
DCTCP	Data Center TCP

List of Figures

List of Tables

1 Introduction

Data Centers and Data Center Networks are gaining more and more importance because of a wide number of applications that rely on them. These applications that are more and more moving to the cloud and going away from our immediate reach of desktops, laptops and servers can serve a number of functions. They might be search applications, file storage and distribution, social networking, email, web servers, gaming and large-scale data-intense computations. For the purpose of supporting the load that these applications bring and the needs that come with them, the focus is driven towards the Data Centers and Data Center Networking.

During the conduct of a study of data centers and data center networking, the focus is driven towards the functionality of the aforementioned applications and their requirements. In this way, conclusions can be drawn for the needs of the data centers and the underlying networks to reach the best results when it comes to data exchange. Data Center Networking is tied to Data Center Architectures and the Network Protocols that are used for the information exchange.

The data center environment being different from that of the general Internet, the necessity for a data center-tailored solution becomes apparent. A very popular protocol used for years in information exchange and the one that functions in a variety of different environments is TCP. Even though TCP as a protocol works in the robust, high speed, low latency environment of the data centers, it is not tailored for it. The solution to the optimization for the specific environment is a data center-tailored protocol, namely DCTCP, that makes use of Explicit Congestion Notification (ECN). While it is difficult to deploy ECN in the wide Internet because of the behaviour of network components that one cannot control, in the data centers, where everything falls under the same ownership, its deployment becomes easier. In this way, ECN can provide the basis for a tailored protocol.

At the same time questions rise about the way this tailored for the data center environment protocol would function outside the controlled conditions of a data center. To be able to answer the behaviour of the protocols, data center conditions and challenges need to be studied. In addition characteristics of the protocol need to be outlined and its conduct in different environments has to be evaluated. This thesis aims on answering some of these questions in a manner that will give an insight on what has been done until now, how and what the conclusions are.

1.1 Research objectives

The main research objective of the thesis is to analyse intra data center transport protocols and especially the data center-tailored protocol DCTCP and its differences from the established protocols. This is an important step in studying the protocol itself but it also builds on the goal of trying to see how the protocol will behave in long distance links. This leads to the answer of the question if this protocol could be used in inter data center transport scenarios.

In addition, importance is put in the way the protocol behaves in congested links and the needs for this new design, since if the protocol is usable outside the data

center environment, it will need to coexist with other protocols, especially TCP.

This research, for the purpose of this thesis is done in an emulated environment with the use of Mininet. To do this it is necessary to have a clear idea about everything that has to do with the environment that is studied. To observe the differences and its characteristics, all the components that are connected with DCTCP are presented.

1.2 Structure of this thesis

The way this study was conducted is tied to the research objectives.

In the beginning, a general overview is given of the environment that is being studied. An overview of the data center environment is concluded and a deeper look on the functionalities of these environments. Through this study the focus is driven to important characteristics and changes that are necessary to further outline the differences between the controlled data center environment and the general Internet.

The importance of transport protocols in the data center networks is made clear and a survey of a number of protocols is included. In a next step, the importance of signalling becomes prominent. A research on signalling is done and an analysis on that level is given. By reaching some conclusions from all the researched data, an evaluation of these protocols is made, based on criteria set by the functionalities themselves.

Lastly, an emulated testbed is used to create a setup for the purpose of getting some information that mirror better how such a scenario would run in the real world. For this purpose the Mininet network emulator is used and further analysed in the specific section of this thesis. Because of the extensive use of this tool for the practical results, Mininet and the difficulties that arose from its use, as well as the environment that it creates, are presented in a thorough way.

2 Data Center Networks

The study of Data Center Transport Protocols has its roots in the expansion and growth of Cloud Computing. According to the definition given by NIST, "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [24] Data Centers and Data Center Networking are connected with this definition because they provide the basis for the services that are provided by the cloud. They are the physical layer of a cloud infrastructure. Data centers comprise of large clusters of servers that are interconnected through switches.

To support the different kind of clouds and their operations and characteristics data centers are built in different ways. One of the main differences from one data center to another can be found in their architecture. Different architectures can also mirror the main services each data center provides and they are a basis for the flexibility of the environment so it can adapt as quickly and cost-effectively as possible to changes in the services and the demands.

2.1 Architectures

Data center architectures have been discussed in a number of studies conducted e.g. [39] [22]. The main architecture that is in use in a data center is a three-tier architecture. This architecture is tree-based and we see in Figure 1 that it involves three layers, namely edge, aggregation and core layer. The top of rack (ToR) switches connect the servers to aggregation switches, which are connected themselves through core switches. The switches that connect the edge and aggregation layers are called edge switches.

With the large amount of traffic that these links get, a number of problems with this basic architecture become prominent and the need to explore some solutions is obvious. The main issues that can be created are link oversubscription, scalability and link failure. To resolve these issues, a number of different topologies that can be used has been explored. The use of a specific architecture is dependent on many variables and it cannot be said that there is only one "right topology" to be used, but the architectures can be compared on some levels and the optimization solutions they strive for can be outlined.

Figure 2 shows a taxonomy of data center architectures according to [22], where it is obvious that there is a big number of topologies that can be used. From these many proposed topologies, priority will be given to some of them that are more commonly referred to and they will be further analysed. These researched architectures can be mainly separated in two categories, fixed and flexible. In the category of fixed architectures, one can distinguish the tree-based and the recursive topologies and the flexible architectures are the ones where the topology can be reconfigured.

[1] *Source of picture:* http://www.slideshare.net/AnkitaMahajan2/introduction-to-data-center-network-architecture,page4

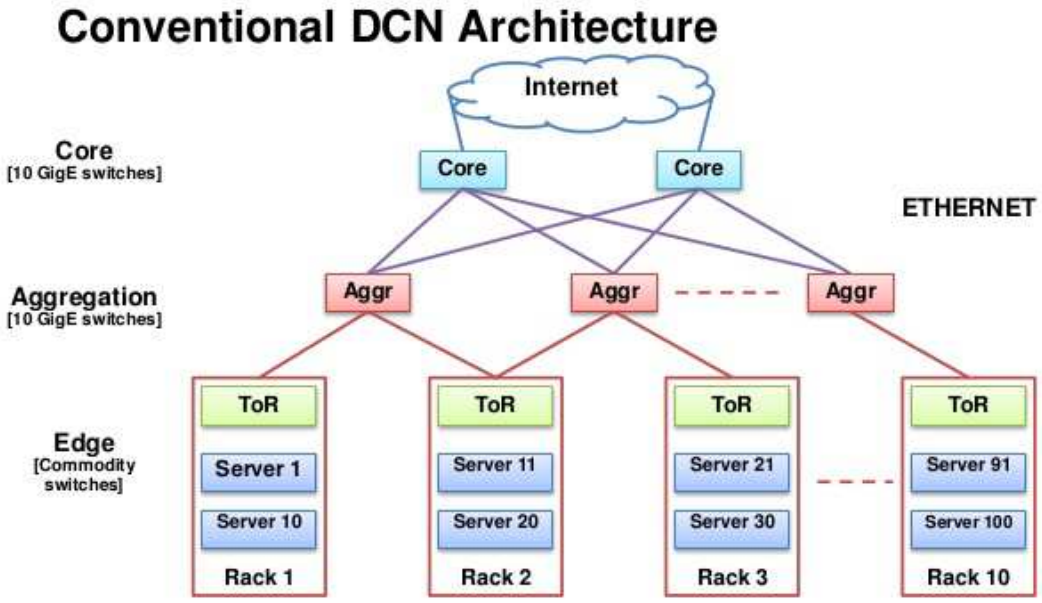

Figure 1: Data Center Networks common architecture
[1]

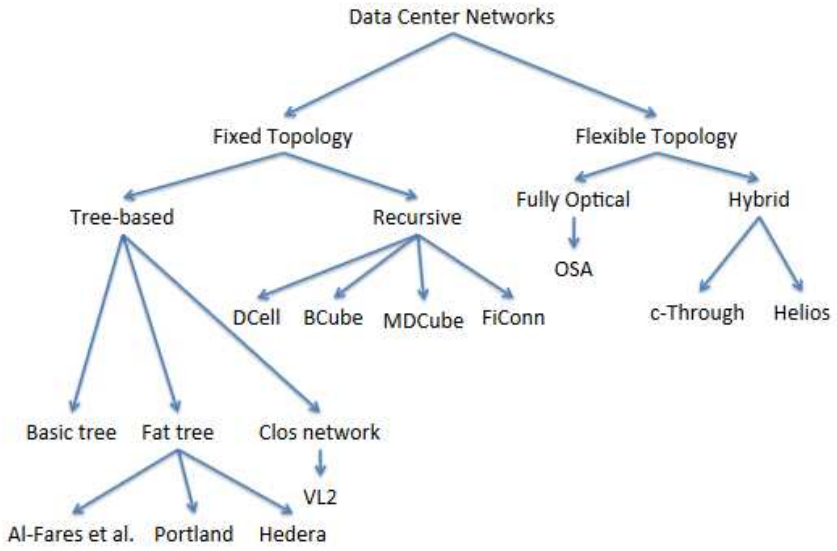

Figure 2: A taxonomy of Data Center Topologies [22, p. 4, fig.1]

Very central in the tree-based technologies are the Clos Network and the Fat-Tree topologies.

The Fat-Tree topology is an extension of the basic tree topology that is so commonly used in data centers. It has three level of switches, namely the core, aggregation and edge switches. A n-ary Fat-Tree topology creates n pods. Each

pod is a structure that contains $(\frac{n}{2})^2$ servers, $\frac{n}{2}$ n-port edge and $\frac{n}{2}$ n-port aggregation switches. The Fat-Tree topology in general contains $\frac{5n^2}{4}$ n-port switches and $\frac{n^3}{4}$servers. The interconnection between the switches and the servers happens by creating a Clos topology and as it is showed in Figure 3.

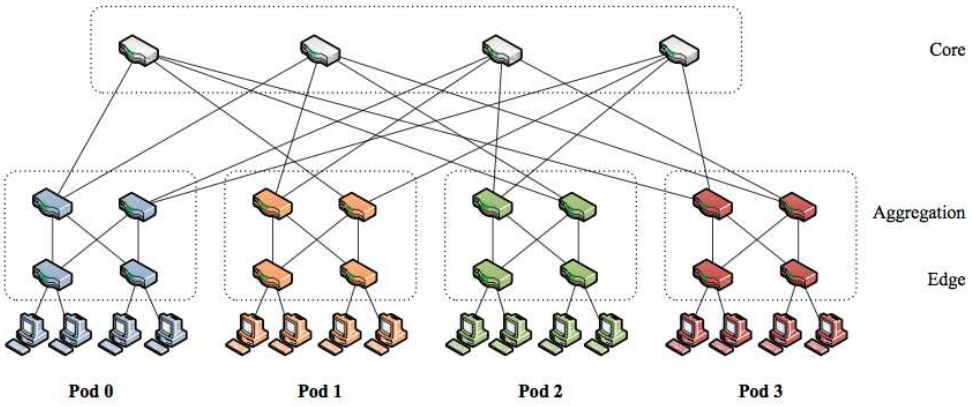

Figure 3: A n-ary Fat-Tree topology with n=4 [28, p. 4, fig. 1]

The rest of the tree-based topologies are created with similar principles. What one can see directly in these cases is the connection of servers to only one of the switches' layers. This is what provides the main difference with the architectures that are part of the recursive topologies group. The servers in the recursive topologies might be interconnected to other servers or to switches of different layers.

Main recursive topologies very commonly used in the research literature and very basic are DCell and BCube. These technologies were initially created for container based data centers, which consist of the data center components fit into a standard shipping container so it is easier to have a data center in a specific location. These technologies apply the principle of building each consequent layer based on an element that consists the basis with a switch and servers.

DCell is the topology that can be seen in Figure 4. Each DCell comprises of elements called $DCell_0$. Each one of these elements consists of n servers and one switch with n ports, with each server connecting to one port of the switch in the same element. Moving from one element to a structure with more elements, each server of a $DCell_0$ is connected to a server in another $DCell_0$. This way every $DCell_0$ connects to another $DCell_0$ with exactly one link. Recursively thus a $DCell_n$ is constructed from $DCell_{n-1}$s. Eventually in a $DCell_n$ each server will have n+1 links to neighbouring $DCell_0$s. It is obvious from the way this topology is constructed that there is a limitation and it is that of the number of NICs on the servers that comprise it.

The basic element $BCube_0$ of BCube is exactly the same as $DCell_0$. The difference between the two topologies is obvious when the higher levels are created. In BCube,

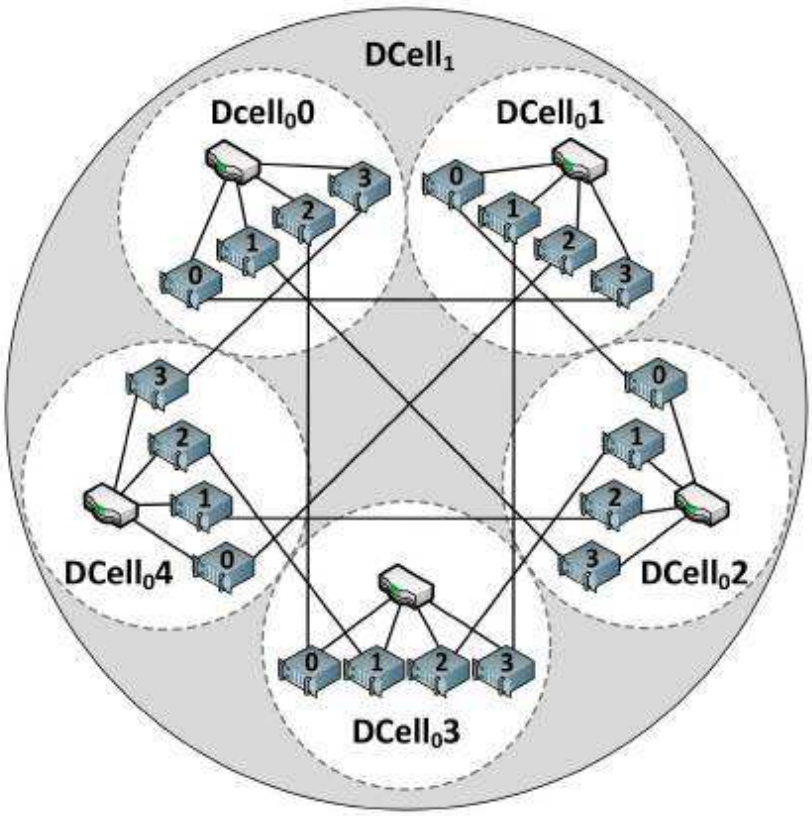

Figure 4: A n-ary DCell topology with n=4 [12, p. 6, fig. 4]

n extra switches are used and they connect each to one server in $BCube_0$, as is seen in Figure 5. In BCube as well the limitation of the NICs of the servers is the same.

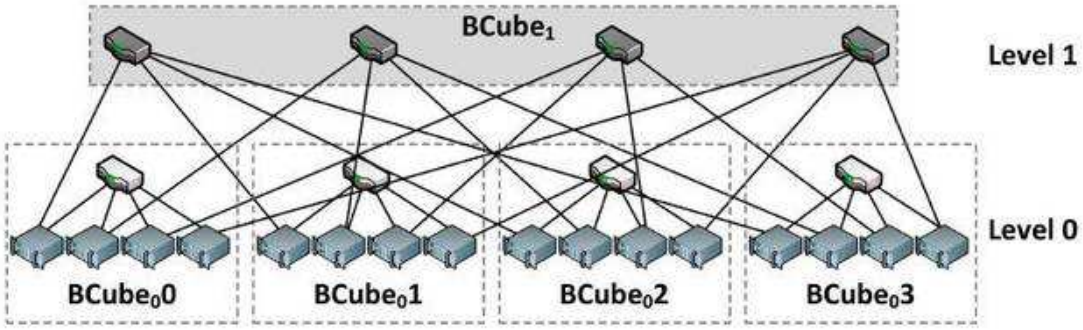

Figure 5: A n-ary BCube topology with n=4 [12, p. 6, fig. 4]

In the category of flexible data center architectures it is worth mentioning the basic principle on which the technologies that comprise it are based. This technology is optical switching. The characteristics of optical switching are used in these topologies by changing during the operation the bandwidth that the optical links can offer and

to which kind of traffic. These topologies are usually hybrid, since they are used together with common electrical switching.

After having outlined these topologies, the levels on which different architectural solutions are usually compared should be mentioned. This can help understand why research usually focuses on some solutions more than others and can set the ground for optimization solutions by seeing what is already out there and what needs to be done in the future.

The comparison of the different technologies can be done on two levels, that of the characteristics and that of the performance.

On the characteristics level what usually gets compared is the number of switches and the number of wires, that are connected to the cost of the topology, the number of servers, which is connected to the scalability of the architecture, the average number of ports on them, otherwise known as degree of the servers, and the diameter, which is the longest of the shortest paths between two servers and has to do with routing efficiency.

On the performance level very important comparing factors are the bandwidth, that has to do with what the topology has to offer for different traffic patterns and the bisection width, which is the minimal number of links to remove to partition a network in two equal parts and is connected to the resiliency against failures of the architecture.

In Tables 1 and 2 a basic comparison that the author of [22] has done is provided and basic benefits of one architecture over another on some levels can be seen. Below k depicts the number of ports in a server, N the total number of servers and n the number of ports in a switch.

	Tree-based Architecture			Recursive Architecture	
	Basic Tree	Fat Tree	Clos Network	DCell	BCube
Degree of Servers	1	1	1	k+1	k+1
Diameter	$2log_{n-1}N$	6	6	$2^{k+1}-1$	log_nN
Nr. of Switches	$\frac{n^2+n+1}{n^3}N$	$\frac{5N}{n}$	$\frac{3}{2}n+\frac{n^2}{4}$	$\frac{N}{n}$	$\frac{N}{n}log_nN$
Nr. of Wires	$\frac{n}{n-1}(N-1)$	$Nlog_{\frac{n}{2}}\frac{N}{2}$	$N+\frac{4N}{n_{ToR}}$	$(\frac{k}{2}+1)N$	$Nlog_nN$
Nr. of Servers	$(n-1)^3$	$\frac{n^3}{4}$	$\frac{n^2}{4}n_{ToR}$	$>=(n+\frac{1}{2})^{2^k}-\frac{1}{2}$ $<=(n+1)^{2^k}-1$	n^{k+1}

Table 1: A comparison of the parameters for different topologies [22, p. 8, Table II]

2.2 Data Center Traffic Characteristics

After having observed the technologies deployed in data centers and having set apart some of their basic characteristics, there is a need to study the traffic patterns of the

	Tree-based Architecture			Recursive Architecture	
	Basic Tree	Fat Tree	Clos Network	DCell	BCube
One-to-one	1	1	1	k+1	k+1
One-to-several	1	1	1	k+1	k+1
One-to-all	1	1	1	k+1	k+1
All-to-all	n	N	$\frac{2N}{n_{ToR}}$	$> \frac{N}{2^k}$	$\frac{n}{n-1}(N-1)$
Bisection width	$\frac{n}{2}$	$\frac{N}{2}$	$\frac{N}{n_{ToR}}$	$> \frac{N}{4log_n N}$	$\frac{N}{2}$

Table 2: A comparison of the performance metrics for different topologies. Bandwidths are expressed as the number of links. "One-to-one" means one arbitrary server to another and "All-to-all" means every server to all the others [22, p. 9, Table IV]

data centers themselves [7] [8]. The data traffic is connected with the traffic inside a data center and the traffic between a data center and the outside Internet.

There are different types of data centers based on the different purposes they serve and the applications they may host. This is directly connected with the kind of traffic each data center has to be designed for. From university and private companies dedicated ones that can contain around thousand servers, to large ones owned by online service providers that might scale up easily to 10K servers, there is a variety of traffic that data centers are demanded to accommodate.

From the different applications that these data centers might mainly host and the services they provide, there is a number of observations that can be made for the traffic patterns and an equal number of conclusions that can be drawn. The basic applications that one has to discern traffic patterns for range from Web services and file storing to data intensive applications like Map-Reduce. Some findings from studies of specific traffic will be mentioned, to give an idea of main patterns.

In research the analysis of 19 commercial data centers with Simple Network Management Protocol (SNMP) traffic patterns leads to interesting findings. All the studied data centers have tree-based topologies. The main conclusions [7] are:

- Most flows have small size and most of them last less than a few hundred of milliseconds. The number of active flows per second is less than 10000 per rack.
- The nature of traffic from a rack is ON-OFF, meaning that packets arrive in groups, and has properties that fit heavy-tailed distributions.
- It was found that link utilizations usually are rather low in all layers but the core.
- In cloud data centers most of the traffic stays in the racks, while for university and private enterprise ones it leaves the rack. While for cloud data centers this is explained as they try to optimize the jobs by placing similar components in

the same racks, it is not certain why the traffic pattern is such in university and private enterprise clouds. one possible explanation is that the placement in these clouds is not as optimized.

- Losses are greater on the aggregation layer than the edge or core layers and occur as a result of momentary spikes rather than high utilization.
- Link utilizations are very based on the time of the day and the day of the week.

In addition to these conclusions, after studying the traffic patterns of network related events in a data center for two months, the following conclusions were drawn [18]:

- Two main patterns comprise the biggest part of traffic in the data center. These are the work-seeks-bandwidth pattern, the traffic of data between servers of a rack, and the scatter-gather pattern, the traffic from map-reduce applications and servers between a cluster.
- Congestion occurs very often in the data centers, especially short-lived congestion.
- Almost 80% of the flows last less than 10 seconds.

Taking into account the conclusions one can safely separate the data center traffic in three main categories. First the small flows traffic, which also consists most of the data center traffic and which has lower data transmission volume, then the small and medium sized flows and last the large updates.

2.3 Challenges in the Data Center Environment

The main goal of research in the field of data center networking is usually how to achieve high energy efficiency in data centers. That is obviously straightforwardly connected with costs and power consumption and utilization. It has been established through research and observation that the servers are not fully utilized and that data centers consume nevertheless huge amounts of energy.

From the analysed traffic patterns one can see the main link utilization and power consumption patterns. Because the data centers that were studied are real ones that are in use, one can generalize these findings for most data centers. To give an opinion about an optimized solution, there is also a necessity of outlining the problems and the challenges that the data center environment itself poses.

Taking into account a typical environment, that in today's data centers is a tree-based architecture, there is a number of challenges that require optimization. As mentioned in research, these challenges include[39]:

- The possibility of adding new components instead of the upgrading of already existing ones.

- The possibility of allocating servers directly to services instead of the physical mapping that is prevalent in the typical environment by mapping an application to specific switches and servers. This also creates the issue of resource allocation, since the data center is not elastic with resource reassignment.
- Every server should be able to connect to every other in the data center using full bandwidth. This does not happen in the typical data centers and the problem of bandwidth bottleneck happens in the aggregation or core switches due to the tree-based nature of the architecture.
- Another issue is the load balance. As it was mentioned earlier, there is an imbalance in the utilization of links in the core and edge layers.
- Because of the load balance and the architecture, issues are created during the failure of a component. The problem is obviously greater if a core switch fails.
- One more issue of the typical architecture is the cost, connected mainly here with the extreme costs of high-end switches in case of cluster expansions.
- Power Consumption is high and inefficient because it is not proportional to utilization and mainly directed from the servers' oversubscription.
- Network Capacity, connected with high bandwidth services being constrained due to server oversubscription ratio.

Taking into account the traffic patterns mentioned, one can realize that data centers' requirements as per the traffic are high burst tolerance, low latency and high throughput. [1, 33]

Having mentioned what we want to tackle in the data center environment and establishing that Ethernet is the established protocol for the data center networks to communicate, there is a need to go one step deeper in the study of the data center environment. In light of this the next step is the study of the data center protocols as they are the basis for the transmission of the data. And the start has to be done with the study of TCP since it is the main transport protocol in the Internet and most incoming traffic in data center environments is subsequently TCP traffic.

3 Transmission Control Protocol

Transmission Control Protocol (TCP) is the main transport protocol of the Internet. In light of this, it is necessary to go deeper in the behaviour of the protocol, since it is an already existing design that can be used in data centers. Because data centers are isolated domains and have their own characteristics and requirements, TCP has to adapt to the changes in this environment.

TCP is the reliable, connection-oriented data transport protocol of the Transport layer of the Open Systems Interconnection model (OSI model). It started as a part of the Transmissions Control Protocol/Internet Protocol (TCP/IP) network architecture to transfer packets with a common protocol between different packet networks and it is the main protocol used to send packets in the Internet.

The fact that the protocol is connection-oriented means that the applications need to establish a connection with each other before the exchange of data. In addition, the protocol is executed only in the end systems and the network elements, e.g. routers, switches, do not process or change anything in the protocol.

A TCP connection provides a full-duplex service, meaning that once a connection has been established, both ends of connection can send and receive data. The connection is also point-to-point, meaning it always connects one sender and one receiver.

The processes communicate via sockets. Each TCP socket is identified by an IP address and a port number. When a process wants to initiate a connection with another process, it behaves as a *Client* and sends a request to the process it wants to establish a connection with, that functions as a *Server*. When the *Server* replies, the *Client* replies back and thus the connection is established. This connection is identified by the pair of sockets, so the two IP addreses and the ports numbers that have been exchanged. The protocol puts a TCP header in the data that is exchanged, creating this way **TCP segments**.

TCP segments have two parts, a header part and a data part, as shown in Figure 6.

In the header one can see the **Source and Destination Port** fields and the **Checksum** field that is used for error check. The **Receive Window** field is used for establishing the number of bytes that a receiver can receive. An also important part that is used in differentiations of the TCP protocol, as it will also be shown later, is the **Flag** field, that has the six bits, URG, ACK, PSH, RST, SYN and FIN, that depending on their status 1 or 0 mean different things depending on the situation.

Two of the fields in the TCP header are the **Sequence Number (SEQ)** and the **Aknowledgement Number (ACK)**, since they are directly connected with the reliability that the protocol provides. This means that by use of these two numbers, TCP makes sure that all packets are delivered across the network, without duplicates and in the right order. With SEQ it makes sure the messages are delivered in order and the receiver by way of sending an ACK acknowledges the arrival of a segment. In case a segment has not been acknowledged, the protocol might need to retransmit

[2] *Image source: Chapter 11, Manual Transmissions, TCP Fundamentals:* http://cse1.net/recaps/11-tcpip.html

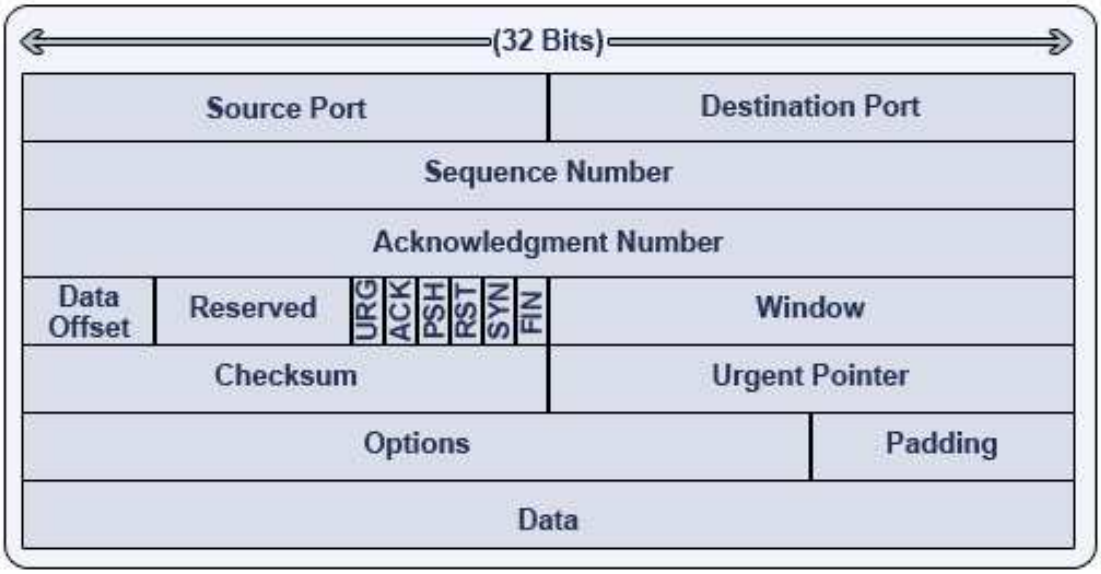

Figure 6: The TCP segment
[2]

that segment.

A client starts a connection establishment to a server by sending a segment with the special flag **SYN**, which means synchronize field. The sequence number in this segment will be established randomly to avoid errors. The server acknowledges this message by sending a segment with the special flag **ACK** and again a random sequence number. The connection is established when the client acknowledges with his own **ACK** segment and the next sequence number of his own message and next acknowledgement number the sequence number of the received message from the server. Figure 7 shows the process.

3.1 Timeout and Retransmission Procedure in TCP

Very important part of the TCP protocol is the retransmission procedure. When a segment is transmitted, a timer is set for that segment. If this timer expires before the segment is acknowledged, then this segment is retransmitted. Because of the different delays of the networks where the segment might be transmitted, the protocol estimates the Round Trip Times (RTT)continuously, since definitely the timers should last longer than the RTT.

RTT is the time for a signal to travel from a destination to a source until the acknowledgement for this signal is received. The way this RTT is estimated is given by the equation

$$EstimatedRTT = (1 - a) * EstimatedRTT + a * SampleRTT \tag{1}$$

where $SampleRTT$ is the measured time for a transmitted segment, $EstimatedRTT$ is the average of the $SampleRTT$s and a is a weighting factor that can be established

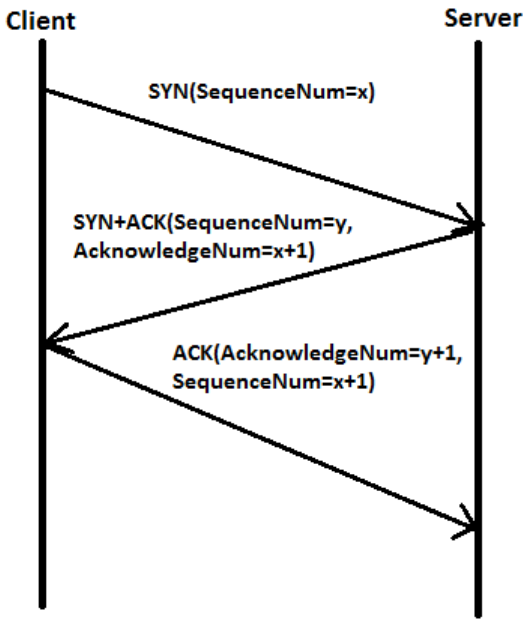

Figure 7: TCP three-way handshake. Visible is also the progression of SYN and ACK numbers

taking values between 0 and 1. The RTT value is an exponential weighted moving average (EWMA) according to the definition of EWMA in statistics. More weight is given to recent samples, which is normal, since recent values mirror better the current congestion. Another important factor for RTT is the deviation. Deviation of RTT is given by

$$DevRTT = (1 - b) * DevRTT + b * |SampleRTT - EstimatedRTT| \quad (2)$$

[20].

Through this procedure a timeout value is established. This value should be bigger or equal than the Estimated RTT, but not much bigger since the bigger this difference is the more delays in retransmission. So the margin between the timeout and the EstimatedRTT should be bigger when there is big deviation between SampleRTT values and smaller when the deviation is smaller. The equation taking into account these facts is given by

$$Timeout = EstimatedRTT + 4 * DevRTT \quad (3)$$

Different TCP approaches can have different ways of establishing this timeout value [20].

In the same category falls Fast Retransmit, which is the procedure that is followed when the transmitter considers the sent segment lost. The way this procedure functions is that when a segment is received with a sequence number bigger than the next sequence number expected, a gap in the data sequence is detected. Because the

receiver has no other way of informing the transmitter, it sends another ACK for the last segment that arrived in correct order. When the sender receives three ACKs for the same data segment, it considers the next segment lost and it retransmits it before this segments timeout, which is called fast retransmit. These procedures are extremely important since they consist the baseline of reliable transmission.

In addition, the protocol uses buffers and windows for the flow control. There is a receive buffer for data that is not yet delivered to the application and a send buffer for data that has not yet been sent. The state of the receive buffer is given by the advertised window field in the header of each segment, which informs the transmitter of the number of available buffers in the receiver. When this window size gets exhausted, the transmitter will stop sending. This avoids slow receiver buffers to overflow by quick transmitters.

Last, error handling and payload are covered with the use of checksums, which ensure that the segments have been sent to the correct address and that the type of the protocol is TCP[20] [21].

Because of the mentioned characteristics and being there when Internet first started expanding, there is no doubt that TCP is the basic protocol used in most networks. For this reason, even if there are new protocols that can be used, they always will have to have compatibility with TCP.

3.2 Congestion control in TCP

One of the most important factors in the way protocols function is congestion. Congestion is bad for all sides involved in a transmission since it can possibly create losses in the links, huge delays, useless retransmits that consume the available bandwidth and loss in throughput and reliability. During the mid 1980s congestion collapses were observed in the networks. Congestion collapse is the situation where in a loaded network every packet is retransmitted many times leading to the buffers in the nodes to be full and the necessity to drop packets. This on itself leads to degradation of the network and sometimes even to timeouts because of the increase of RTTs [26]. In this situation, the need for the protocol to adjust the sending rate based on the rate of incoming acknowledgements and loss rate became apparent.

There are two main ways to do congestion control. End-to-end-congestion control is one of them and it is when the network components do not contribute to the congestion control. Every conclusion of congestion comes as a result of observations in the terminal systems. On the other hand, there is congestion control with the help of the network, where the network components provide clear feedback to the terminals for the existence of congestion.

TCP uses end-to-end congestion control, since IP does not provide feedback about congestion to the components of the network. The way that congestion control works in TCP is that every TCP transmitter has to half the rate of transmission when congestion is observed. If on the other hand, there is no apparent congestion, the sender gradually increases the rate of transmission.

In more detail, a congestion window(CWND) parameter is used in all terminals so the congestion situation can be observed and to regulate the rate at which the

sender can transmit. This parameter is maintained by the sender and it is calculated by the estimation of congestion between sender and receiver. The value of CWND is set and increased during every RTT when there is no observation of loss or it follows the slow start pattern in the beginning of each connection.

The existence of congestion in a link is observed by the timeout that might happen or by the reception of 3 ACKs for the same segment, which shows loss. On the other hand, in the case that there is no congestion in the network, ACKs will be normally received, so TCP will assume that the network is functioning properly, in which case the CWND will be increased according to the rate with which the acknowledgements arrive. AIMD algorithm also varies the congestion window.

The TCP congestion control algorithm has three components: (1) Additive Increase/Multiplicative Decrease (AIMD), (2) Slow Start and (3) Reaction to timeouts.

The basic idea behind AIMD is the transmitter to decrease the rate of transmission based on the size of the CWND. The mechanism behind this is that a sender increases under normal circumstances the rate of sending in an additive manner, but when a loss is detected, the transmission rate is halved in a multiplicative way. The phase of the additional increase is also called Congestion Avoidance (CA). This phase provides the traditional "sawtooth pattern" in long TCP connections, as we can see in Figure 8.

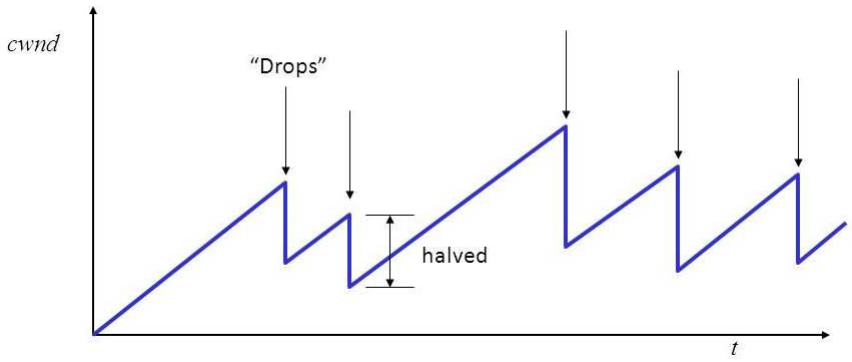

Figure 8: Additive Increase Multiplicative Decrease CWND pattern [3]

When a TCP connection starts, the CWND is set to 1 MSS. The Maximum Segment Size (MSS) is a parameter in the TCP header options field that specifies the biggest amount of bytes that a host can receive in a TCP segment. In this initial state, the protocol assumes that there is the whole bandwidth available so it increases exponentially the CWND size every RTT until a loss event occurs. When the loss event occurs, as it was described beforehand, TCP enters in the AIMD phase, where the CWND increase is done now linearly, hence the slow start phase.

[3] *Image source: CS144 An Introduction to Computer Networks Flow Control, Congestion Control and the size of Router Buffers Section Nick McKeown Professor of Electrical:*http://slideplayer.com/slide/6928333/

In reality, TCP congestion control reacts differently to a loss that occurs because of a timeout and a loss that is observed because of three consecutive ACKs. In the case of the ACKs, this is the way it reacts. In the case of a timeout, it enters the slow start phase and the CWND size is increased until it reaches a threshold, in which case the protocol enters in a CA phase.

Depending on the behaviour during a congestion event, there are different versions of TCP algorithms and obviously different reactions. As an example, TCP Reno, a TCP version, cancels the slow start phase after a triple ACK event, a process that is called Fast Recovery.

The point of TCP protocol congestion control is to proactively probe the capacity of the network path, in lack of other explicit information. The negative aspect of this method is that the protocol can react only when buffers are full and packet losses happen and do not prevent these losses.

3.3 Challenges in Data Centers for TCP

Because of the reason TCP is the main established protocol, consequently in data centers the main traffic consists of TCP segments. Data center environment being different from the environment of the Internet though makes the TCP face some challenges in data center networks, since it is a protocol mainly created for the Internet.

The type of traffic in data centers was analysed before, as well as the requirements of the data center environment. These characteristics are directly connected with the main challenges of the data center environment. Outlining these challenges in more detail, they can be summarized in **TCP Incast**, **TCP Outcast**, **Queue buildup** and **Buffer pressure**.

One of the most known problems that is created is that of the TCP Incast. It has been defined as the pathological behaviour of TCP that results in gross under-utilization of the link capacity in various many-to-one communication patterns[11]. This will happen when the switch interface will be overflowed with data from different flows concurrently. The possibility of the flows synchronizing is quite big, since the time boundaries are very small when talking of applications using the partition/aggregation model and that is exactly where the issue might be created by having synchronous transmissions of data. Thus, incast is created in the switch port connected to the aggregator where data from different flows arrives at the same time and there is extreme congestion. In more detail, when a request is sent from the aggregator to the layers below, the layer below will split the task between the nodes. When the task is completed, all nodes will send the results to the aggregation layer and because the task was equally distributed the data from all nodes will arrive almost simultaneously. The more nodes are added to the model, the bigger the problem that is created. Figure 9 shows a TCP incast scenario.

"TCP Outcast is the name of the situation when a lot of flows and a few flows arrive at different input ports of a switch, but compete for the same bottleneck output port and the small set loses on the throughput."[31] The main factors that cause this situation are the many-to-one patterns and the use of drop-tail queues

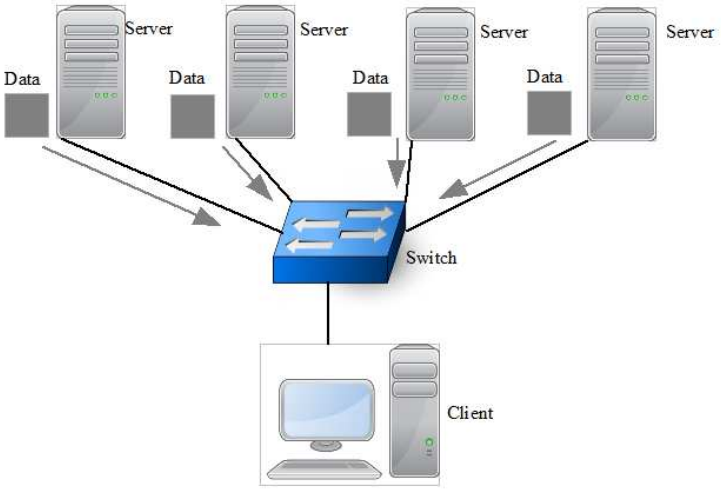

Figure 9: Depiction of TCP incast

in the switches, which is quite common in data center networks. These drop-tail queues, because of the packet drops and frequent timeouts, cause the high latencies of small flows.

Queue buildup is the situation when the packets of small flows start building queues behind the packets of large flows. This happens when large and small flows use the same queue and because large flows are usually background traffic that occupy most of the buffer space.

This impairment is a direct consequence of queue buildup, since the buffer space available is reduced because of the background traffic large flows. The result is the same as that of incast, with the difference that here synchronization of the flows is not necessary.

As a result of the aforementioned impairments, a lot of research has been done and a lot of different solutions are available. One of the most popular solutions is changes on the transport protocols or new transport protocols or designs that attempt to solve the issues. Some of these solution rely on TCP or change some of the features of it, while others are designs that use already established algorithms or other protocols. In the next section, a survey of these solutions will be presented, so it is obvious what has been done in research and what innovation it brings and what challenge each solution tackles.

3.4 TCP Improvements for Data Center Networks

There is a number of improvements studied for the purpose of using TCP in the data center environment. Each of these improvements changes some parts of the protocol to tackle one or more of the mentioned challenges in data centers.

One of the designs is **TCP Timeout Improvements** [35] [33]. In a previous part the way TCP times the segments and the retransmission process were described. Based on observations and analysis the retransmission timeout (RTO) is estimated and set for TCP connections. Taking into consideration that the average round-trip

time(RTT) in networks like the Internet is around hundreds of milliseconds gives usually the value 200ms to the RTO. The data center being different though and RTT on the order of microseconds, the value of RTO turns out to be quite bigger than that of RTT. Through simulations and measurements in real clusters the innovations that this TCP variant brings is the improvement of RTO by enabling timeouts with microsecond granularity, which also makes necessary the existence of high resolution timers. An addition to this approach is connected to delayed ACKs, which is the condition when the receiver aknowledges every two received segments. Because with such fine RTO timers, spurious retransmissions might occur, the solution is to either reduce the delayed ACKs' timeout period to a few microseconds or to disable the timeouts completely. The goal of this approach is to improve the overall throughput, reduce the impact of packet loss, and to effectively avoid incast collapse. While the TCP throughput is improved and it improves on TCP incast, the approach reacts to packet loss without actually avoiding it and it does not solve the other challenges of TCP in data centers. Another characteristic of this approach is that it does not need many modifications of the TCP.

Another improvement is **Incast congestion Control TCP (ICTCP)** [37]. This approach is an incast congestion control scheme for TCP on the receiver side that regulates the receiving window. The reaction happens on the receiver side based on the fact that the bandwidth there is a signal for congestion control and that the receiver side can adjust according to the requirements of both the application and the link. ICTCP tries to avoid packet loss before incast congestion happens by adjusting the receive window of each connection based on the total bandwidth of the incoming traffic. This is done by having each flow estimating the increase of throughput and by measuring the actual throughput of a connection. Through experimental results one can see that the approach tackles the incast challenge on data center networks, as it was its original goal.

In continuation there is the approach of **Multipath TCP (MPTCP)**[32]. Multipath TCP is a version of the TCP protocol which uses multiple paths by spreading data from a single TCP connection to multiple subflows that can take different paths in the network. Each subflow has its own congestion window and subflows with larger windows increase them faster than the ones with smaller windows. The use of MPTCP is negotiated upon connection but the applications do not need to know that MPTCP is used. The protocol achieves increased network utilization since underutilized and idle links can be explored. It also provides fairer allocation of capacities to flows and robustness by using multiple paths and by avoiding to send traffic to congested ones. MPTCP has been proved through simulations and experiments to do better in today's data centers, since it is compatible also with the aforementioned architectures of FatTree and BCube, even though fairness and performance have also to do with the number of subflows. On occasions it ourperforms TCP, but they both are the same when it comes to being aggresive.

3.5 Other Transport Designs

In addition, there is a number of protocols tailored for the data center environment. Changes in the TCP protocol is not the only way to impove designs for the data center environment. Some others are protocols designed from scratch, sometimes based on already existing techniques and put together to perform better in data centers. There are a number of algorithmic designs as well, focusing mainly on the improvement of performance in the data center. These designs can be based on queueing heuristics or be standalone designs. In continuation some of these designs are presented.

Rate Control Protocol (RCP) is a protocol that is mainly created to keep flow completion times low. It achieves this goal by the routers "stamping" a rate $R(t)$ of every link to the packets that pass through. The sender receives these values and gets informed of the slowest rate or bottleneck and adjusts the rate that it should be using, avoiding this way slow start and managing to complete flows quickly and in fairness [13].

D^3 ("Deadline-Driven Delivery") is a control protocol specifically designed for data centers. It is not based on TCP and for its implementation an endhost-based stack and a router that supports the protocol were created. It is also taken for granted the existence of flow deadline and size when the flow transmission begins. On a first phase, when the flow is initiated, the source and each router along the path request a rate that is carried as information in the header. Through the ACKs of the flows, the source gets informed about this requested rate and sends the data with the smallest from the requested ones, while at the same time sends the request for the next RTT. The routers assign rate requests by trying to satisfy as many flow deadlines as possible, while the remainig flows will send at least a packet with a header so they can get the request for next RTT through. The goals of the protocol are to maximize the application throughput, to accommodate flow bursts and high network utilization. Compared to TCP, the protocol achieves similar performance for long flows and shorter queues. On what can be considered a negative aspect of D^3, the protocol calls for changes in all components of the network and the transmission [36].

Preemptive Distributed Quick (PDQ) is a layer that can be added between the IP and transport layers and is designed for quick flow completion by meeting the flow deadlines. The way the protocol works is by taking into account that the information about the flows exists in their headers through the PDQ switches. These switches read the information and are the ones responsible to inform the transmitters about the rates that are going to be used, while at the same time they perform "dynamic decentralized scheduling by building on traditional scheduling techniques like Earliest Deadline First (EDL) and Shortest Job First (SJB). Research has been done as well on Multipath-PDQ (M-PDQ), with which a flow is transmitted on multiple paths. The same principle is followed while the paused subflows shift their load to the transmitting ones with the least load. Through simulations it was found that the protocol performs better comparatively to TCP, D^3 and RCP as the number of flows increases and it has better flow completion times [17].

pFabric is a data center transport design that has on the basis of its design the idea to separately handle flow scheduling and rate control. Another core characteristic of the protocol design is the fact that each segment has on its header a priority number. This number is set on end hosts on the transport layer and the switches decide which packets to accept and which to drop based on this number. The switches also maintain queues with the arrived packets and their metadata for easy access if a packet is dropped because the buffers were full. The rate control is a simple version of the TCP rate control, which means all flows start at line rate and decrease their sending rate only if they see high and persistent loss. At the same time, there are no more complex mechanisms that TCP has. The protocol was evaluated through simulations and it reaches almost ideal flow completion times and it handles incast like other protocols, but depending on the scenario its fairness might suffer [4].

PASE is a transport framework that combines existing transport strategies, like **P**rioritization, **A**rbitration and **S**elf-adjusting **E**ndpoints. It adopts the self-adjusting endpoints from TCP strategies, where senders make their rate decisions based on the observed network conditions. From the queueing algorithms it adopts the arbitration, where a part of the network allocates rates to each flow and it uses in-network prioritization like in pFabric, where switches schedule and drop packets based on some priority number [25].

Hedera is a flow scheduling system. Its general design has to do with giving large flows good paths with the use of placement algorithms by estimating their demands. After the estimation, the design uses the Simulated Annealing technique to compute the flow paths and having a switch per receiver rather than per flow. The goal of the scheme is to maximize the aggregate network utilization without impact on active flows.Through simulations it has been found that the design outperforms some of the specifically designed for data centers protocols [14].

DeTail is a design that uses cross-layer mechanisms. The goals of the design are to reduce packet losses and retransmissions, prioritize latency-sensitive flows and balance traffic across multiple paths. The design detects congestion at lower network layers and finds different paths with less congestion to destination, leading this way to the routing of the flows. The lossless fabric deployed ensures that losses occur only because of hardware errors or failures [38].

FastPass is a data center network architecture that makes use of centralized arbitration to determine the time at which packets should be transmitted and which paths should be used. With FastPass there is no congestion at the switches and the endpoints can transmit at wire-speed since the architecture uses a timeslot allocation and a path assignment algorithm. The goal is to send the packets in a way that will avoid congestion and make use of full paths dealing with problems that might arise in the network a priori [30].

This chapter focuses on the TCP protocol, it being the main protocol used for data transmission. The changes in TCP to tailor it for the data center environment are also brought forward and a number of other transport designs that tackle the issues of transmission in the data center environment.

4 Queue Management and Explicit Congestion Notification

Having studied already the cause of congestion and the end-to-end methods that the protocols use to react to it, we are going to study in more depth the network based methods that are deployed for congestion avoidance and reaction to congestion.[9] [5]

4.1 Queue Management

Traditionally, what happens in the queues of the routers involves dropping packets from the queues if a maximum length of each queue is reached with the incoming packet. This mechanism is called "drop-tail", since the last arriving packet, "the tail", is dropped if the queue is full. Drop-tail mechanism has the drawback that it can lead to the occupation of the queues from specific flows or connections only, so it does not treat fairly the incoming packets. Another very important drawback is that the devices do not give feedback until the queues are really full so losses can be avoided and this leads to a throughput reduction and to problematic *bursty* traffic with delays and reduction in performance.

Other mechanisms exist, like dropping a random packet if the full is queue instead of the last arrived, or drop the first packet in the queue. In the same way though, these mechanisms do not solve the problem of early feedback and the maintenance of small queues that can solve the issue of degrading throughput.

There has been discussion already about the congestion avoidance mechanisms that TCP deploys to avoid the collapse of the network. With the growth of Internet during the years though, it has become obvious that network devices need also to cooperate with the already existing mechanisms on the edges against congestion collapse so the protocols can perform better.

The congestion control in the network devices, like routers, is done in two main ways, with scheduling algorithms and with queue management. Scheduling algorithms regulate the way in which arriving packets are handled and forwarded and allocate bandwidth, while queue management algorithms handle the queues of the network devices when these queues are created because the incoming packet rate is bigger than the outgoing.

The reasons for the queues to be managed derives from the need to reduce queue delay and to evenly distribute packet losses between flows.

4.1.1 Active Queue Management

Intuitively, the solution to the queues problem is for the network devices to manage the buffers before queues become full by controlling their lengths or the time the packets spend there. Such an approach is called Active Queue Management (AQM)[5]. AQM mechanisms lead to less packets being dropped in routers, since keeping the queues small more packets can be absorbed in queues if they come in bursts. At the same time the perceived delays get reduced and by buffer space being available for all

flows, the incoming traffic management becomes more fair. The deployment of AQM mechanisms leads thus to tackling an important issue, that of the Internet latency.

AQM mechanisms are used to control the queue sizes from the scheduling algorithms. Depending on the priorities the scheduling algorithms want to give and the flow or class queues that are created from these algorithms, AQM can contribute to these queues by reducing delays and minimizing the losses. Additionally, instead of packet drops, AQM might use ECN to mark packets during congestion events or even introduce delays to packets. This way, the end points get informed about congestion beforehand, so losses and delay can be avoided. The way AQM is used should not be protocol-oriented or application-oriented, but to provide a solution to the current measured conditions of congestion that might occur in a network.

4.1.2 Random Early Detection (RED)

It becomes apparent that the AQM mechanism that might be used depends on a number of factors that have to do with network conditions and other measurements. This was not always the case though. When AQM was first introduced, the main mechanism that was proposed was Random Early Detection (RED).

RED is an algorithm that functions in two phases. The first phase is the estimation of the queue size making use of an exponential weighted moving average. This avoids extreme increases of the queue size by bursty traffic. In the second phase, RED takes the decision if a packet drop is going to occur or not. This is done by comparing the measured average queue size *avgq* against two threshold values, *minimum* and *maximum*, that are chosen based on the desired *avgq*. If *avgq* is smaller than *minimum*, no packets are marked. If it is larger than *maximum*, all arriving packets are marked. When the value is between the two thresholds, the packets are marked with a probability that varies linearly from 0 to *maxp*, which are functions of *avgq*. To understand this in a simple manner, one would say that packets are dropped if the queue has been mostly full in the "recent" past. [15]

RED is not the only AQM algorithm that is deployed in the networks. Two other very popular algorithms are Controlled Delay (CoDel)[27] and Proportional Integral controller Enhanced (PIE) [29].

4.1.3 Controlled Delay (CoDel)

In the case of CoDel, it has already been used a lot in many implementations, amongst them in Linux. CoDel is different from other AQMs on some points. One of them is that it treats "good" and "bad" queues differently, with "bad" being the queues that keep buffers full. To distinguish these queues, CoDel uses the sojourn times of the packets by tracking the local queue minimum. The algorithm then reacts with the goal of minimal delay and maximal utilization when the delay is bigger than a set target delay. The way CoDel reacts is by dropping packets from the queue and maintaining the number of the dropped packets setting the time for the next drop until the delay falls under target delay again. In this case CoDel exits the dropping state. CoDel algorithm has been used as a basis for other mechanisms or just by itself.

4.1.4 Proportional Integral controller Enhanced (PIE)

In the case of PIE the image of the delay during the whole transmission is used and not only the current delay. As it is mentioned in [29], "similar to RED, PIE randomly drops an incoming packet at the onset of the congestion. The congestion detection, however, is based on the queueing latency instead of the queue length like RED. Furthermore, PIE also uses the derivative (rate of change) of the queueing latency to help determine congestion levels and an appropriate response. The design parameters of PIE are chosen via control theory stability analysis. While these parameters can be fixed to work in various traffic conditions, they could be made self-tuning to optimize system performance". In case of supporting ECN, the algorithms mark the packets rather than dropping them.

4.2 Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) is a network-layer mechanism that is used for notifying of existing congestion. On the network devices side, when ECN is used, it makes use of AQM mechanism. Instead of dropping packets when a congestion event happens, AQM can set a Congestion Experience (CE) codepoint in the IP header of the packet and is translated by the protocol as congestion indication. At the same time, delays due to retransmissions are being avoided, since the packet is not actually dropped.

In the IP header, there is the 2 bits long ECN field, that includes the possible codepoints 00, 01, 10 and 11. If the codepoints are set 01, 10 by the senders, this is an indication of ECN-Capable Transport (ECT), which means that the sender and the receiver are capable of using ECN. If the codepoints are not set, so 00, there is an indication that ECN is not being used and if both codepoints are set by the router, 11, it means that congestion was experienced. If a router is congested it can use congestion notification when it sees these bits set. If these bits are not set, the router will drop the packet instead of marking it, since it will mean that it cannot be processed as an ECN packet by the endpoints.

If a CE packet is received, the protocol should react to it, usually once per window of data and the packet is not dropped, but transmitted normally. This CE codepoint is set by the router if it receives a packet with set ECN codepoints. In the case this packet is erroneously dropped, the end nodes react to congestion as if it was an actual dropped packet, since there is no information that it was the CE packet.

4.2.1 ECN and TCP

The TCP protocol functions with ECN by negotiating the protocol during the connection setup. There are two flags added in the *Reserved* field of the TCP header that indicate use of ECN, the *ECN-Echo* (ECE) that informs for the reception of a CE packet, and the *Congestion Window Reduced* (CWR) that informs that the window size was reduced, as it is shown in Figure 10. These flags are used for signalling between TCP endpoints.

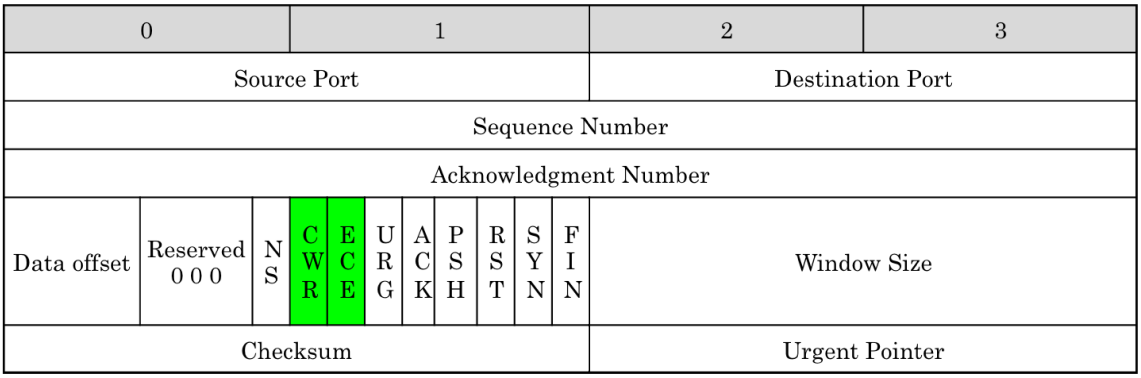

Figure 10: The TCP header with use of ECN flags [4]

During the negotiation phase, if a host receives an ECN-setup SYN packet, it may respond with an ECN-setup SYN-ACK packet if it agrees to the use of ECN. This however, does not compel the host to set ECT, but to respond accordingly to the reception of CE packets. A host can only use ECT only if it has sent one ECN-setup SYN or SYN-ACK packet.

The sequence of events in a TCP connection that makes use of ECN is as follows:

- Sender sends ECT packet to indicate ECN support.
- Network device detects congestion and sets ECT codepoint. CE is also set when ECT in the IP header is detected and the packet is forwarded.
- When the CE packet is received, ECE is set in the next TCP ACK packet that is sent to the sender.
- When the sender receives the ACK packet reacts in the same way that would react to a dropped packet.
- CWR flag is set by the sender to acknowledge the reaction to the ECE packet received.

It is important to note here that the reaction to congestion should be once per series of CE packets or drops from the same data window. If a retransmitted packet is dropped though, a new congestion event is being experienced. At the same time, CWR flag should be set only on the first transmitted data packet, even though it is in response to a series of received ECE ACK packets.

After recognizing congestion as one of the main challenges in data transmission, this chapter focuses on the deployed methods that tackle congestion. This is a necessary step between all the transport protocols that can be used in the data center environment and DCTCP, since the DCTCP algorithm uses ECN feedback.

[4] *Image source:* http://bocloud.blogspot.fi/2013/04/tcp-flags-part-2.html

5 Data Center TCP (DCTCP)

Previously the TCP based algorithms and other protocols tailored for the data center environment were studied. In continuation, a study of DCTCP will be done, it being by far the most popular design for data center environment.

5.1 DCTCP in depth

Data Center TCP (DCTCP) is a TCP variant specifically designed for data centers, with the goal of achieving in the different from wide area data center environment high burst tolerance, low latency and high throughput. The protocol has been broadly used in research as the main protocol for data centers and even deployed, which makes it very interesting for the specific purposes of this study.

DCTCP functions by reacting to congestion according to the extent of congestion experienced. For this purpose, the end hosts that use the protocol need to receive feedback from the switches about the experienced congestion. This is easy to achieve since most modern switches are ECN capable, meaning they can provide explicit feedback so end hosts can react.

The algorithm includes three parts. The first part is an AQM mechanism based on the marking configurable parameter K. Upon a packet arrival, if the queue occupancy is bigger than K, then the packet is marked with CE to indicate congestion. Through a series of experimentation in NS-2 (network simulator) and also through the results from the testbed in [1], there are different values that are proposed for K. These proposed thresholds are K = 20 packets for line rates of 1Gbps and K = 65 packets for line rates of 10Gbps. This is connected to the burstiness of traffic experienced in these links and the fact that to achieve 100% of its throughput, K has been found that needs to be at least 17% of the bandwidth delay product.

The second part of the algorithm has to do with the reaction to the experienced congestion. It was mentioned that for TCP the receiver sets the ECE in response to a series of ECE ACKs until the reception of a CWR packet. In DCTCP, every packet is ECE marked if it has a CE codepoint mark. In the case of delayed ACKs, the receiver uses the two state machine shown in Figure 11 to decide if it will set ECN-Echo bit.

On the third part of the algorithm, "the sender maintains an estimate of the fraction of packets that are marked, called a, which is updated once for every window of data (roughly one RTT) as follows:

$$a \leftarrow (1 - g) * a + g * F \tag{4}$$

where F is the fraction of packets that were marked in the last window of data, and $0 < g < 1$ is the weight given to new samples against the past in the estimation of a. Given that the sender receives marks for every packet when the queue length is higher than K and does not receive any marks when the queue length is below K, the equation implies that a estimates the probability that the queue size is greater than K. Essentially, a close to 0 indicates low, and a close to 1 indicates high levels of congestion." [1, p. 5]

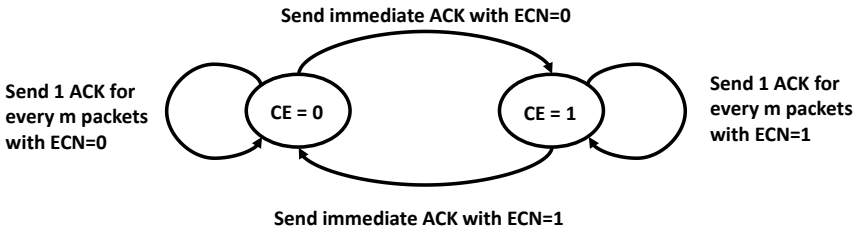

Figure 11: The two state ACK generation state machine. The states correspond to whether the last receved packet was marked with the CE codepoint or not. [1, p. 5]

It was also mentioned how TCP reacts to a congestion event. For DCTCP, everything is the same except for the new window size. In DCTCP, the new value of the CWND is given by

$$cwnd \leftarrow cwnd * (1 - a/2) \tag{5}$$

[1, p. 6], meaning that the CWND is not halved, but reduced according to the amount of congestion encountered. When the congestion is really high, DCTCP behaviour becomes similar to TCP.

The results from the use of protocol in simulations in research have been so that the protocol outperforms TCP in data center traffic scenarios. At the same time though, because of the way DCTCP changes the CWND, the protocol's convergence time is longer than that of TCP. In spite of this for the data center environment and the traffic patterns one can observe there, the difference is not considered important.

The way the protocol functions leads to it tackling the challenges that were mentioned for the data center environment. Because of the threshold value K, there are no queueing delays and it even reacts to the buffer problem since the queues do not grow large. Because of the early marking, buffer overflows and timeouts are prevented.

Because of the wide use of the protocol, there is a number of changes that are made or proposed for it that lead to improvements of its performance.

DCTCP modifications

Some modifications that are made in the protocol portray the experience from its use and solutions to impairments found from the practicalities of the protocol. Here the changes that are introduced have to do with a number of variables that

inform the sender side about the actual amount of congestion encountered. Thus, the boolean variable DCTCP.CE that is stored in the Transmission Control Block is introduced and is used when sending an ACK, the ECE flag MUST be set if and only if DCTCP.CE is true. Another value that is introduced is DCTCP.Alpha, which is the estimation of the fraction of bytes that encountered congestion. The window is reduced based on this parameter, which is in turn updated by the estimation gain and the congested bytes during the previous RTT. The gain should be set depending on the implementation and the congested bytes are set by counting the marked and sent bytes on the observation window of choice. DCTCP only works if both ends, sender and receiver, are DCTCP-capable. Another crucial point is that the use of DCTCP is explicit inside data centers or exchanges that can be named to exist inside data center networks. In addition, this version of DCTCP is implemented on Microsoft Windows Server 2012. [1, 2, 6]

5.2 DCTCP-based Data Center protocols

Next some protocols that are based on DCTCP will be mentioned to additionally outline the importance of DCTCP for the data center environment. The fact that there are already designs that use DCTCP as a basis and improve upon it further shows the protocol's importance.

First, there is **Double-Threshold DCTCP (DT-DCTCP)** [10]. DT-DCTCP improves on the original DCTCP by using two parameters K1 and K2 instead of the threshold K. K1 is to start ECN marking in advance, informing that there might be congestion in the network so the CE codepoint should be set. If the queue length gets smaller than the value K2, the CE codepoint stops being set.

Next the **High-bandwidth Ultra-Low Latency architecture (HULL)** design that uses DCTCP mechanisms [3]. HULL is an architecture that has as a goal the achievement of ultra-low latency and high bandwidth utilization. The HULL architecture consists of three main components: DCTCP congestion control, phantom queues (PQ), and packet pacing. The PQ mechanism is practically a counter that sets ECN marks based on link utilization and not on queue occupancy, achieving low latency because of the signaling before queueing occurs. HULL reacts to these ECN marks using DCTCP's mechanism. In the HULL architecture, pacing takes place in the hardware after the last source of bursty transmission and only the packets that belong to large flows are paced, since otherwise it exist the danger of introducing end-to-end latency while decreasing network latency.

Further on **Deadline-Aware DCTCP (D^2TCP)** [34]. D^2TCP is a TCP-based data center network protocol that aims at achieving high bandwidth for background flows while meeting the Online Data Intensive (OLDI) applications' deadlines. At the same time, it requires no changes to the switches. D^2TCP) changes the congestion window size through a gamma-correction function based on deadline information and encountered congestion. If the flows have no deadlines, then D^2TCP) functions as DCTCP.

Finally, **DCTCP with Weighted Random Early Detection (WRED)** [19]. WRED ensures that flows do not starve when coexisting with flows from other, more

aggressive variants. There are two changes proposed to the classical DCTCP. First, the fraction of marked packets to be updated with every acknowledgment packet received and second, decrease of the congestion window whenever an ECN-Echo is received.

6 Analysis

We studied extensively the data center environment and the challenges of this environment that also highlight the differences between the data center networks and the broad Internet. In addition we studied the protocols that are being used and can be used depending on the needs of the network. Very important parts are the characteristics of the TCP and DCTCP protocols, since they are most extensively researched and used.

TCP is so established and so broadly used that this fact is not going to change soon in the networking scene. At the same time through the study of new protocols and algorithms in the data center environment one is made aware of a number of positive results for the networks because of changes in already existing protocols or development of new algorithms. From the comparison of these other designs with TCP in the controlled data center environment one can specify as well the changes that make the algorithms perform better.

Narrowing down to specific results, research has shown that higher throughput is achieved, losses are minimized with the use of marking and lower occupancies are achieved in the buffers minimizing the queueing delay in the case of DCTCP. DCTCP being the most researched protocol for data center environments and at the same time being easy to deploy, the necessity to research its performance on different long distance and loss scenarios immerses itself. This is as a result of the fact that networks would have better performance if the protocols would achieve the same results as DCTCP does in data center environment in the wider Internet.

At the same time, because most incoming traffic in data centers is TCP traffic, the need to study the coexistence of TCP and DCTCP in a link is also important. This is helpful both for the coexistence of TCP and DCTCP traffic in the data center networks and for their coexistence in case DCTCP is used as a transport protocol in the wide Internet as well.

One of the solutions for the use of DCTCP in the internet was provided by the authors in [19], where they provide a number of modifications for the algorithm and the AQM design used that lead to the coexistence of TCP and DCTCP traffic in links without having starvation.

The main difference is the use of a dual AQM design, where packets share one queue, but the traffic is classified based on the ECN capabilities. The authors "propose to standardize an ECN signal that signals congestion immediately, allowing the end hosts to distinguish between smoothed and immediate congestion notification." From the results in simulations running different scenarios of different numbers of long-lived DCTCP and TCP flows, it is proposed in [19] that with specific changes to have more instantaneous and accurate ECN feedback is the actual way to deploy gradually DCTCP in the Internet. The majority of research lately focuses on ECN for the deployment of DCTCP in the internet and the results are promising.

6.1 Tools used in study

For the purpose of studying the scenarios, it is necessary to deploy TCP and DCTCP in a link and see the traffic behaviour. In addition, it is beneficial to study different scenarios with different parameters so we see the protocols' characteristics in a variety of situations. The parameters that were considered that can change were the RTT, the number of nodes, losses in the links and the possibility of sending packets with different transport protocols, since each of these changes give us a different network scenario.

Because of the number of parameters that need to change and test the performance in such diverse scenarios and at the same time because of the necessity to mimic the behaviour of real networks, network emulation was chosen as the best technique to study the traffic.

Network emulation uses virtual test networks to measure the performance of applications by giving the opportunity to change different characteristics of networks like RTT (latency), packet loss, jitter and others. Figure 12 shows a network using lightweight virtualization. Network emulation differs from simulation in the fact that in simulations mainly mathematical and network models are applied, while emulators test by changing the network setup itself. In the process of simulation the main characteristics are taken into account to model the behaviour of a network. Through emulation the goal is to test this behaviour of a system. At the same time especially in networking, network emulation makes it easier to test with a number of switches and servers that are required for testbeds but might not be possible to have as actual hardware because of costs or difficulties in changing the setup.

6.2 Mininet

For the study of the behaviour of TCP, TCP with ECN marking and DCTCP in the scope of this thesis, Mininet network emulator was used to conduct the traffic observation. DCTCP in Mininet is used in the paper [16] as an example for reproducibility of network research in the emulator.

Mininet[6] is a container-based network emulator used to test Software Defined Networking(SDN) applications that uses Python API (Application Programming Interface) to write the different testbed scenarios and compiled C code for the emulation. The emulator can run a number of hardware components in a single Linux kernel. Mininet uses virtualization to make a system look like a real network. A Mininet host behaves like a real host. The packets are exchanged and processed by what looks like real Ethernet links, switches and routers. The difference is that these links and switches are created using software and are not real hardware, but they behave the same way.

Mininet is very helpful for a number of reasons. It can run the same programs

[5] *A mininet tutorial:* https://github.com/mininet/mininet/wiki/SIGCOMM-2014-Tutorial:-Teaching-Computer-Networking-with-Mininet

[6] *Introduction to Mininet:* https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

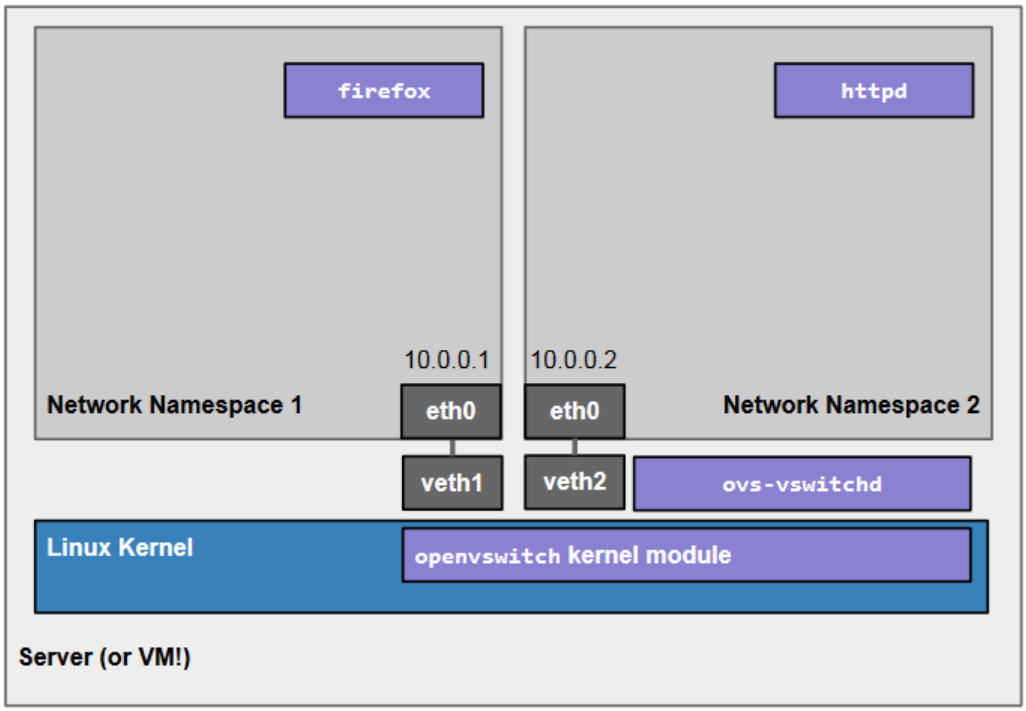

Figure 12: A very simple network using lightweight virtualization [5]

that run on Linux and the emulator can run everywhere, like a laptop, a server or a VM. With the use of Python scripts it is easy to create custom topologies and it is an ongoing open source project with a very active developer community that update the software often and receive feedback. At the same time Mininet uses OpenFlow to program the switches and this makes it easy to customize packet forwarding and transfer the scripts to hardware OpenFlow switches. OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network[23]. Through a number of simulations in Mininet running the same scenarios as in testbeds, it is obvious that Mininet approximates the hardware very closely. Figure 13 [16, p. 6, Fig. 6] shows how results from the same topology run in Mininet and hardware are similar. This practically means that we can safely use Mininet as an emulator with a variety of scenarios.

On the other hand the usage of Mininet imposes some limitations. Since the emulator runs on a single kernel, appropriate resource allocation needs to be done amongst the switches and hosts and the parameters of each emulated topology. At the same time the OpenFlow controller needs to be customized depending on the needs by the user of the software. Last, the environment is isolated from the LAN and the Internet and the timing is based on real-time, meaning that faster than real-time results are difficult to emulate.

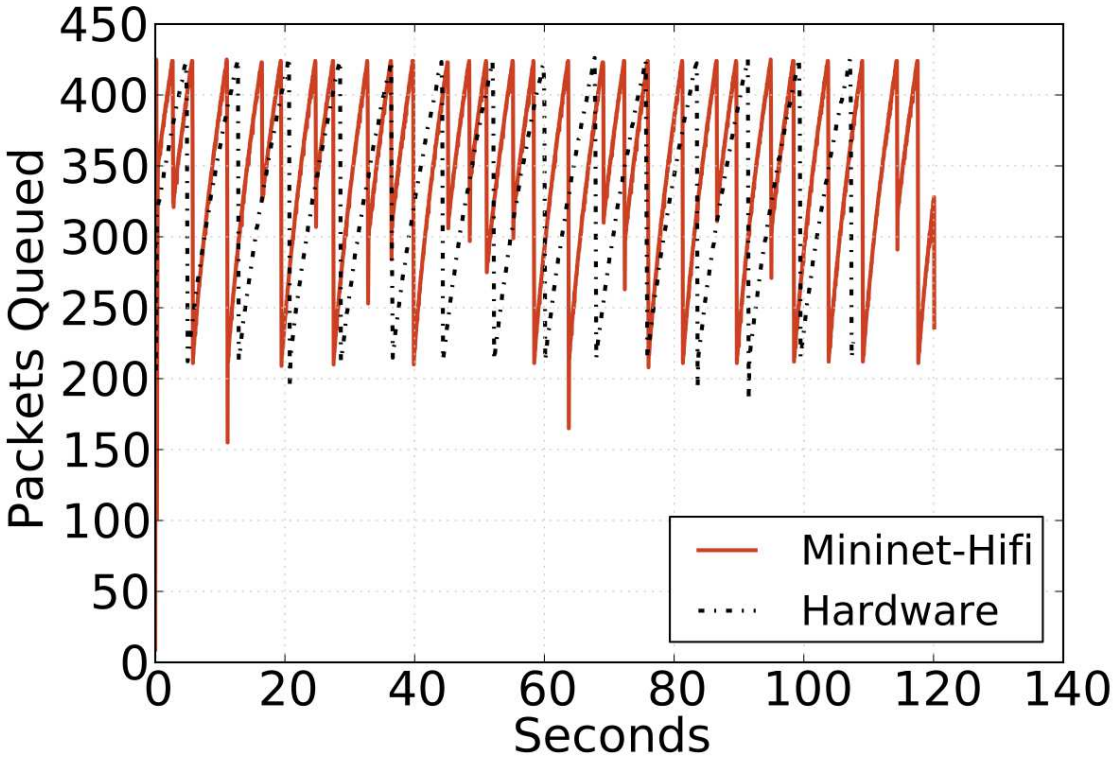

Figure 13: TCP queue occupancy in Mininet and hardware.

6.3 Setup

Emulator Mininet provides the environment to test TCP, TCP with ECN and DCTCP. For this purpose again the environment was based on already existing DCTCP tests in Mininet.[7] Since Mininet is Linux based, the operating system to run the different scenarios is Ubuntu Linux. In the course of creating the environment for running the scenarios there was a number of difficulties that we faced. Because DCTCP was not yet a protocol option in the Linux kernel at the time of the setup, a DCTCP patched kernel was used. This kernel was a patched version of kernel v.3.2 that runs on Ubuntu 12.04.5 LTS [8]. The kernel installation was not completed as the given kernel was not being installed properly on the created environment. Due to a number of incompatibilities during the initial setup and because latest versions of Linux kernels got released that have DCTCP as an enhancement of TCP congestion control, in the final setup Ubuntu 14.04.4 LTS with kernel v.4.2 was used. DCTCP is enabled as a congestion control version in this release with the command $sysctl\ -w\ net.ipv4.tcp\_congestion\_control = dctcp$. The installation was done in

[7] *"Mininet system-level tests, benchmarks, and performance monitoring":* https://github.com/mininet/mininet-tests

[8] *A kernel patch provided by the University of Stanford:* https://github.com/mininet/mininet-tests/blob/master/dctcp/0001-Updated-DCTCP-patch-for-3.2-kernels.patch

a virtual machine and for the environment was given 20Gb of RAM memory and 2 cores with two threads each.

In continuation, different topologies were tried in Mininet. For the purpose of studying the traffic two very simple topologies were chosen as shown in Figures 16 and 17.

The topology of Figure 14 includes a switch and 3 hosts connected to it, but the number of hosts may vary depending on the results we want to see. h1 functions as a receiver and the link h1-s1 as the bottleneck link. h2 and h3 are the sender hosts.

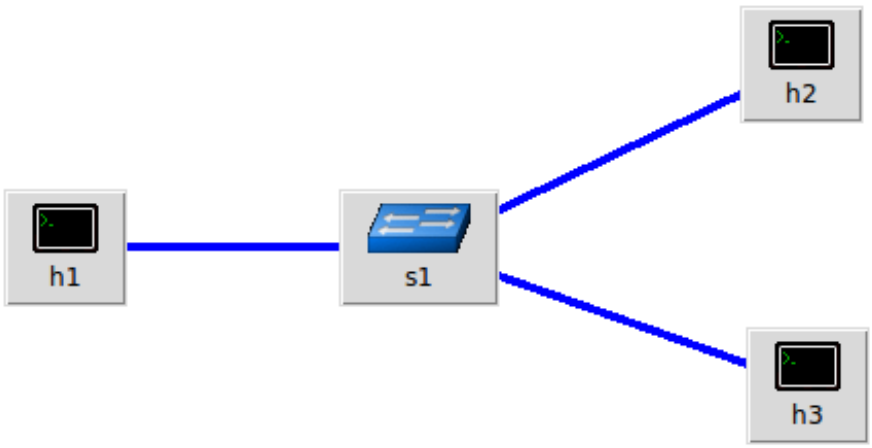

Figure 14: One of the two basic topologies used in Mininet. Depiction with MiniEdit.

The topology of Figure 15 includes two switches connected with each other and each of them is connected to a varying number of hosts. In its basic form each switch is connected to two hosts. The hosts connected to one switch function as receivers, so in the Figure example h1 and h2 connected to s1 function as receivers and h3 and h4 connected to s2 function as senders. The bottleneck in this topology is created in the link between the two switches.

ECN in Linux is implemented using RED. The RED parameters are set to maintain 20 packets threshold on DCTCP. Mininet achieves best performance and similar to hardware when the resources are used proportionally. This means that with links of 10GB the results are not accurate and they can be questionable with links of 1Gb as well. The best solution for this situation is for Mininet to use links of 100MBps to approximate the results that we would have from the use of hardware.

With the topologies mentioned we can run a number of scenarios. The main goal of the emulations is to see the behaviour of each protocol and their variations and how the protocols behave when they coexist in the link.

For the different scenarios a number of parameters changes. The link behaviour was studied with both mentioned topologies and the number of hosts that send traffic to the receivers was varied. At the same time the bandwidth of the links was also

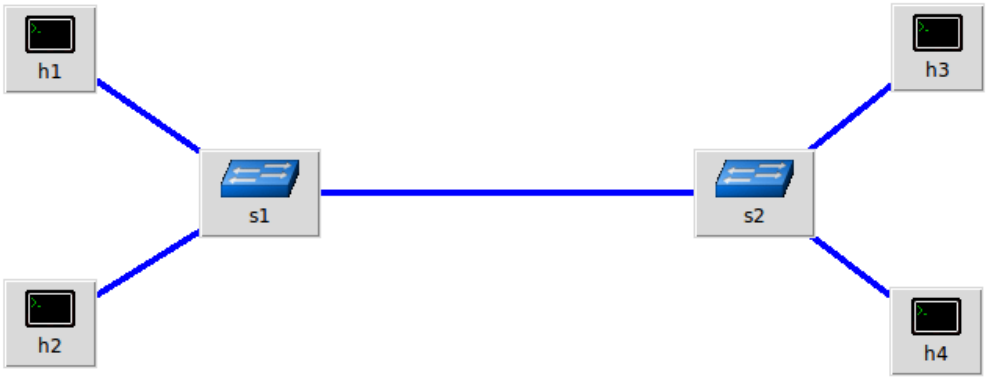

Figure 15: One of the two basic topologies used in Mininet. Depiction with MiniEdit.

changed. In the following figures a number of results is shown from the use of the topology of Figures 14 and 15 and the figures depict the CWND and the queue occupancy of the link when the bandwidth and the number of hosts changes. The differences with the increasing number of hosts from 3 to 100 are made obvious through these depictions and it is also beneficial to see how Mininet behaves with bandwidth of 1GBps, based on the mentioned limitations as well, so the results from links of 1000MBps are shown.

6.4 Evaluation

For the different scenarios the results are presented below. At the same time the constants used for the scenarios that are presented are as follows. The RED parameters used are: *minimum*:30000, *maximum*:35000, packet size of 1500 bytes and marking probability 1. For ECN we have the same values since ECN in Linux is implemented using RED. At the same time the delay was configured to be 0.05ms. The traffic generation is done with *iperf*, which can create TCP streams and functions as a bandwidth measurement tool for the network these packets travel in.

6.4.1 3 hosts,100 MBps

Figure 16 is a CWND against time graph for a link of 100MBps that Mininet can emulate with big approximation to reality and with 3 hosts. As it was mentioned, Linux uses RED congestion control, so here we can see the behaviour of TCP Linux. Because the two links from the two sender hosts are the same, the figure shows the sum of the CWND so we have a more encompassing image. We can see a peak around 8000 kB for these hosts and the rest of the time the CWND values seem to be around 400 and lower. At the same time we can see the results from running exactly the same topology with TCP with ECN congestion control and DCTCP. In the case of ECN TCP, as we can see in Figure 17, the values that CWND takes are

much lower, with the sum of them reaching hardly and in very rare instances like second 15 the peak of 360kB. At the same time the CWND image is much smoother for ECN TCP traffic with increases and decreases that have peaks from 20kB or less up to 100kB in average. In Figure 18, the image that we get from the same topology but with DCTCP traffic this time is very similar to that of Figure 17. This makes sense in the results since DCTCP uses ECN.

Already by running with a small number of hosts we can start seeing the differences between the protocols. Through the images of the CWNDs we can tell for the TCP traffic that CWND starts growing as packets flow in the link. When the peak is reached in the graph the protocol reacts to the congestion experienced. Then the protocol enters the congestion avoidance phase and thus we have the specific image in the graph that suggests the AIMD and CA phases.

Contrary to the way TCP reacts we can see the much smoother reaction in the CWND of DCTCP. The image of the CWND with DCTCP has this shape because of the marking of the packets that mirror the congestion in the last window of data and the adjustment of the CWND according to this marking, which means that the protocol does not enter the AIMD phase like TCP, but reacts by adjusting the CWND according to the marking. Then it informs for the reduction with the CWR parameter. Because of this we can see exactly the importance of the feedback and ECN in the link.

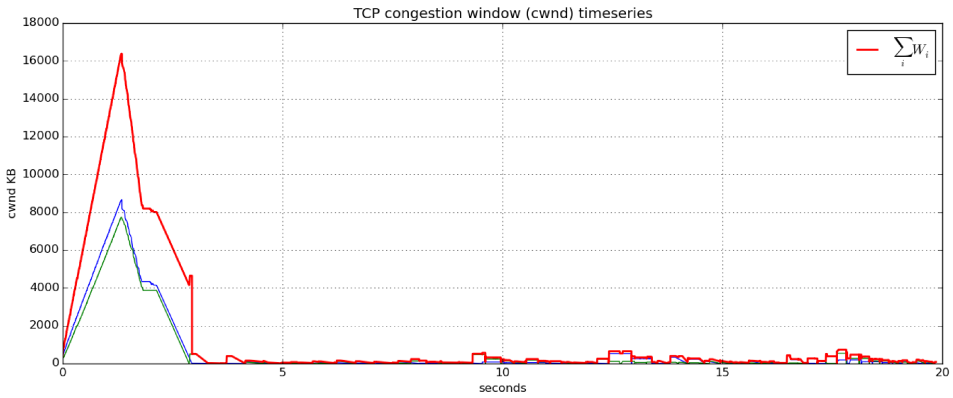

Figure 16: CWND with bandwidth of 100MBps and 3 hosts for TCP traffic

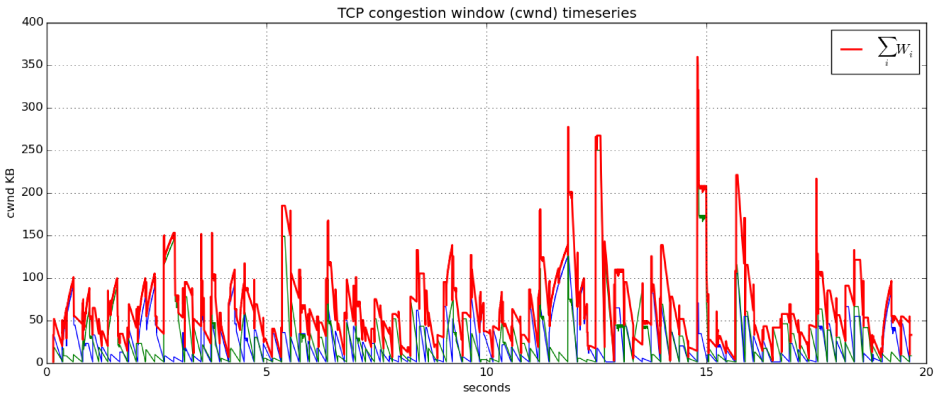

Figure 17: CWND with bandwidth of 100MBps and 3 hosts for ECN TCP traffic

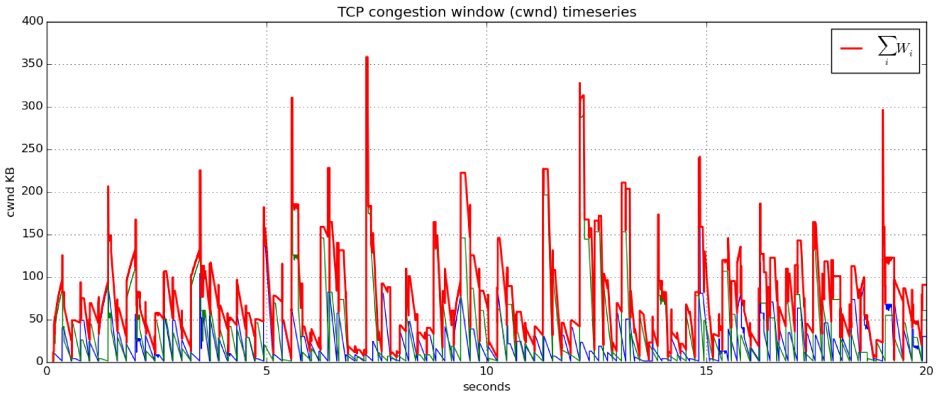

Figure 18: CWND with bandwidth of 100MBps and 3 hosts for DCTCP traffic

The most interesting part of the results lies in the comparison of the queue occupancies with the use of the different protocols in the same topology. Figure 19 shows the queue occupancy with the use of TCP. We can see a big peak in the beginning of the connection and then from the 10th second to the 20th it shows that the queue occupancy is maintained quite high, with values that reach many times peaks of around 200 packets. At the same time, using the same scale, the big difference from the use of ECN is made obvious by Figure 20, where we see the queue occupancy is kept at minimal. Figure 21 shows the queue occupancy for DCTCP, that is kept very low and around 20 packets at its peak, which tells as well that DCTCP is tuned according to the parameters given by the authors in [1]. The images were on purpose done in different scale so this difference is obvious.

With maintaining the queue occupancy low in DCTCP, we have lower queueing delay which affects in turn the smoother image of the DCTCP CWND, since we avoid losses and drops of packets keeping the queues in the marking threshold.

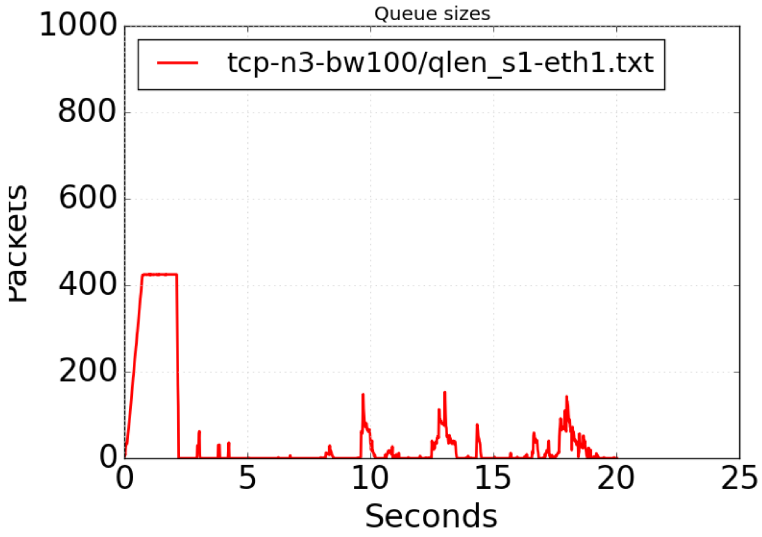

Figure 19: Queue occupancy with bandwidth of 100MBps and 3 hosts for TCP traffic

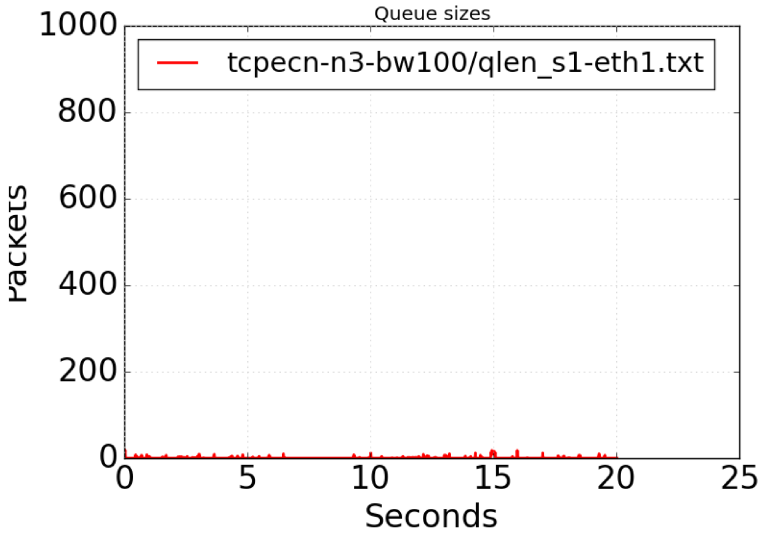

Figure 20: Queue occupancy with bandwidth of 100MBps and 3 hosts for ECN TCP traffic

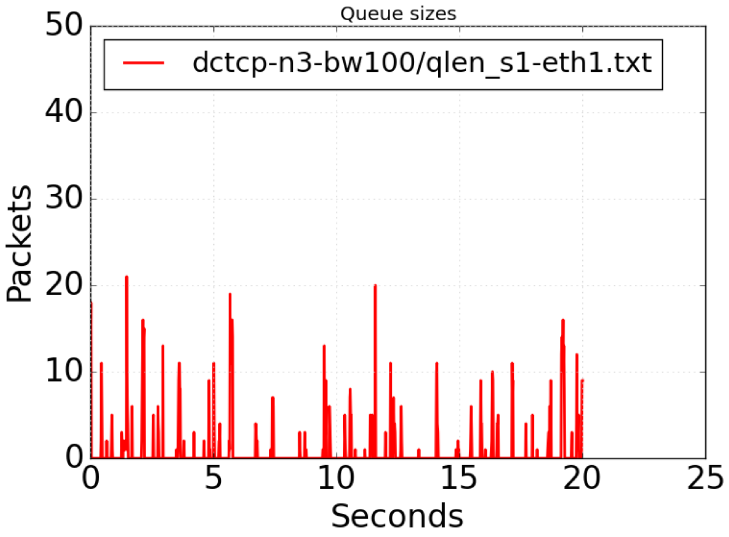

Figure 21: Queue occupancy with bandwidth of 100MBps and 3 hosts for DCTCP traffic

6.4.2 3 hosts,1000 MBps

In continuation, by changing the link bandwidth, but maintaining the number of hosts at 3, we can see some interesting results. Figure 22 shows the CWND in this case with the use of TCP and what is interesting is the very smooth line after the 5th second. We believe that the CWND is like this because of the big speed of the link and the small number of hosts. After the 5th second it appears that there was no congestion indication, while until that moment the window has reached even a 15000kB peak. Figures 23 and 24 show respectively the CWND from the same topology but with the use of ECN TCP and DCTCP. Again we see much smoother curves that reach hardly around 500kB at specific times.

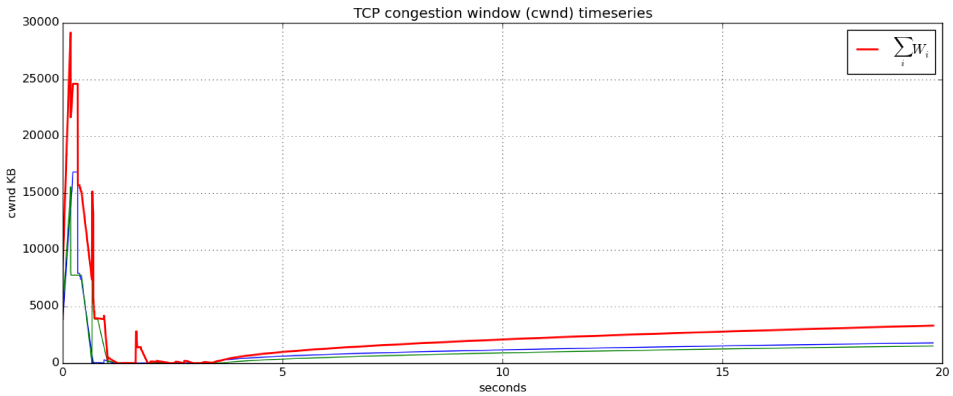

Figure 22: CWND with bandwidth of 1000MBps and 3 hosts for TCP traffic

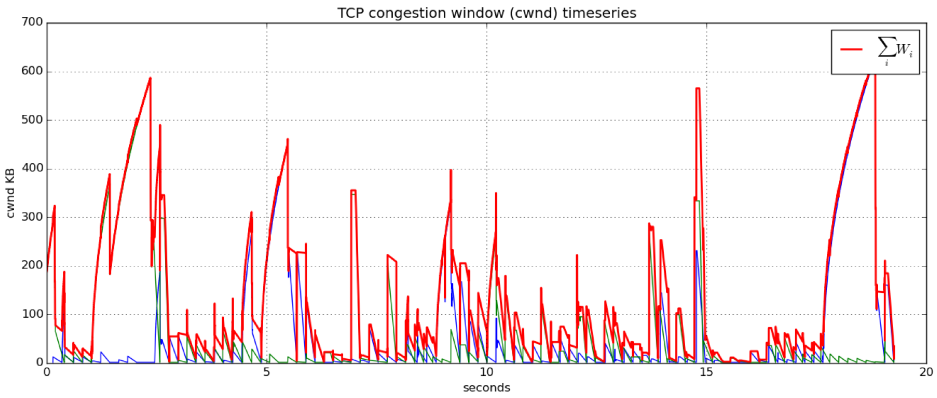

Figure 23: CWND with bandwidth of 1000MBps and 3 hosts for ECN TCP traffic

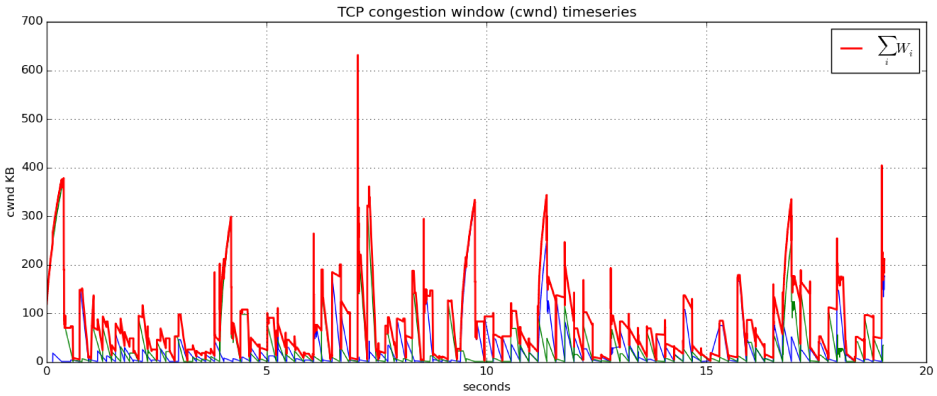

Figure 24: CWND with bandwidth of 1000MBps and 3 hosts for DCTCP traffic

Figures 25, 26, 27 show the queue occupancy in the case of TCP, ECN TCP and DCTCP respectively. Again we can see much smaller queue occupancy in the case of ECN TCP and DCTCP. If what we said about the congestion of the links being low in this case is true, this case is not very interesting since the queues will be mostly empty or will contain a small number of packets. Figure 25 proves this as except from the initial stage the queue occupancy is kept low there as well, especially from the 5th second onwards.

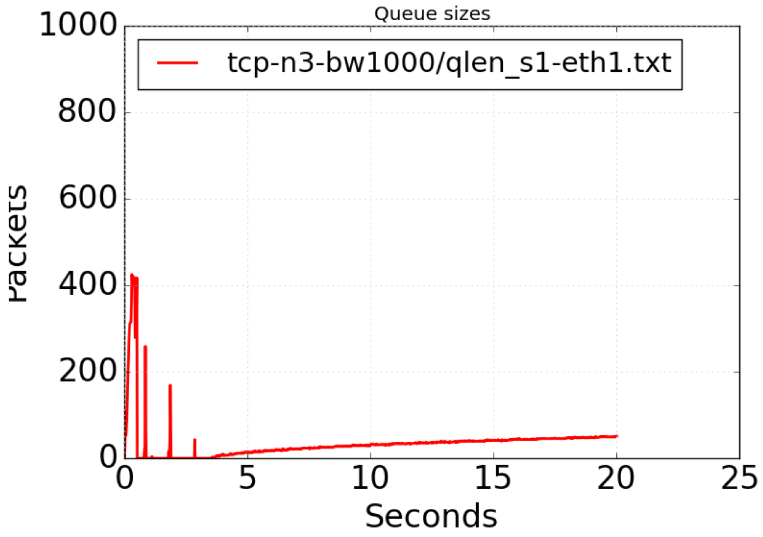

Figure 25: Queue occupancy with bandwidth of 1000MBps and 3 hosts for TCP traffic

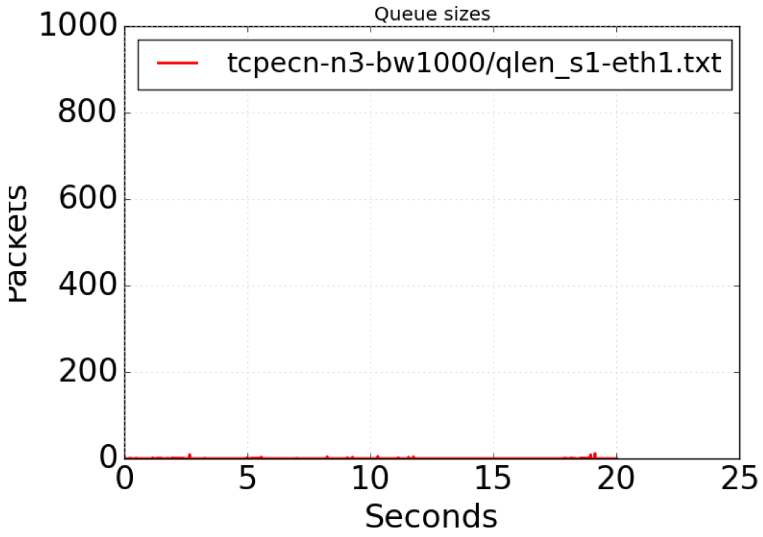

Figure 26: Queue occupancy with bandwidth of 1000MBps and 3 hosts for ECN TCP traffic

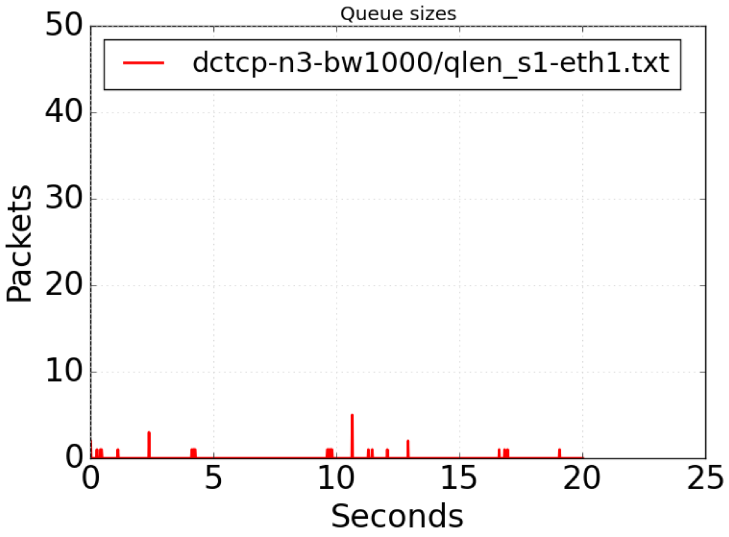

Figure 27: Queue occupancy with bandwidth of 1000MBps and 3 hosts for DCTCP traffic

6.4.3 100 hosts,100 MBps

A very interesting case that can show the benefits of DCTCP and ECN is the case where the number of hosts increases a lot (100 in our case) and the link bandwidth remains at 100MBps, giving thus space for congestion to happen. In this case Figure 28 shows the CWND with the use of TCP. Because of 100 links, the sum of the CWND was depicted, so we can have a general image. We can see that the CWND takes quite high values and we can see quite clearly as well the "sawtooth" behaviour of TCP, especially after the 5th second. At the same time, ECN TCP as depicted in Figure 29 has a smoother behaviour and the CWND is kept at lower values. For DCTCP and Figure 30 again we can see the CWND maintaining lower values compared to TCP, as the sum of the CWND from the 100 links is still kept mainly under 1000kB while for TCP that is a value lower than the average that is around 1500kB.

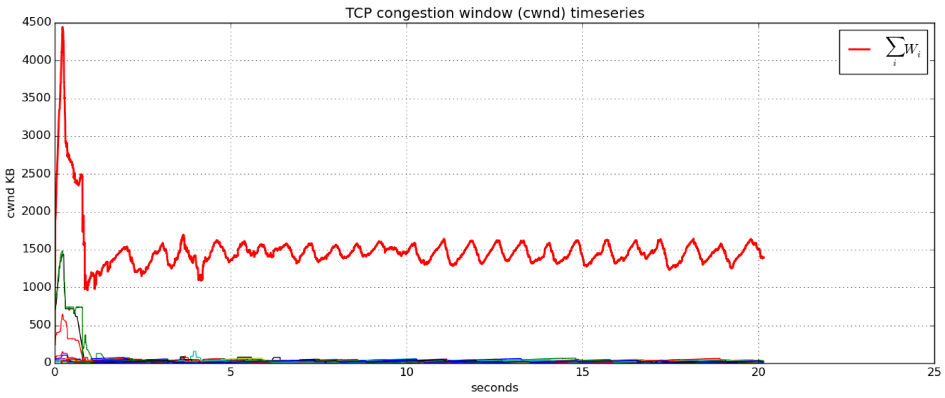

Figure 28: CWND with bandwidth of 100MBps and 100 hosts for TCP traffic

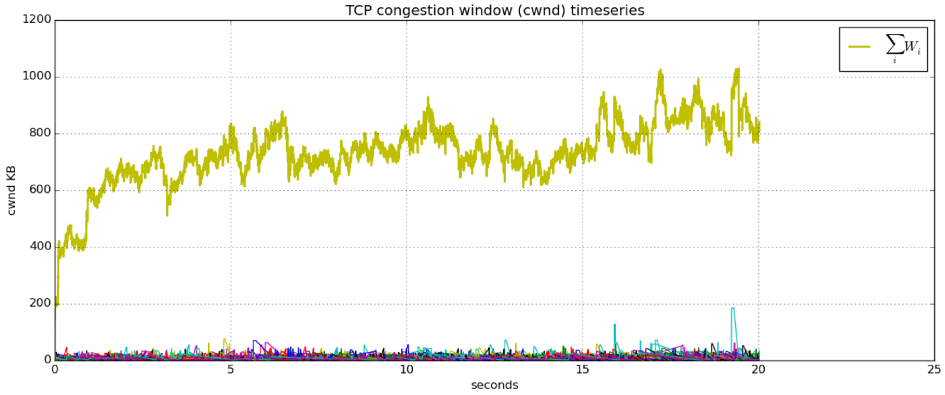

Figure 29: CWND with bandwidth of 100MBps and 100 hosts for ECN TCP traffic

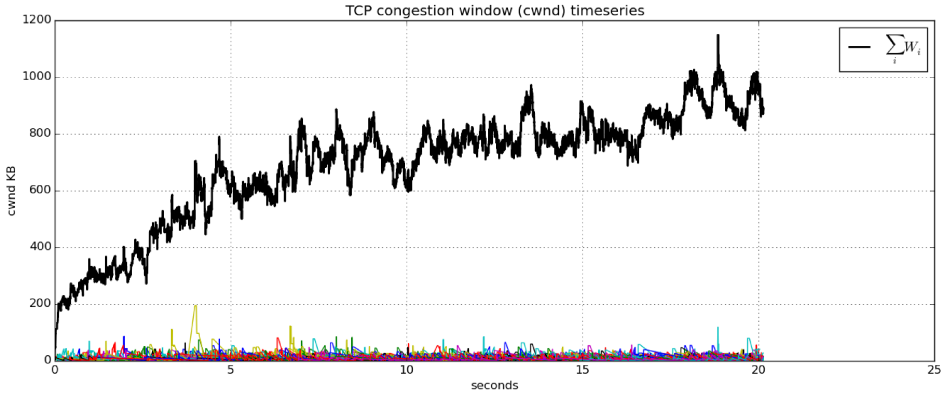

Figure 30: CWND with bandwidth of 100MBps and 100 hosts for DCTCP traffic

Very interesting are the results that have to do with the queue occupancy with the same topology. In Figure 31 it is shown clearly the high queue occupancy that

TCP keeps at all times during the traffic exchange. Comparing that to the queue occupancy of ECN TCP in Figure 32 that is in the same scale, we see the very big difference between the congestion algorithms. At the same time Figure 33 shows that with the use of DCTCP even when we have such a large number of hosts and small links with congestion, the queue occupancy is kept extremely low and it does not get bigger than 20 packets most of the time. When it does we have obvious drops and that is why the image of the CWND is not as smooth for DCTCP when we have such a big number of links.

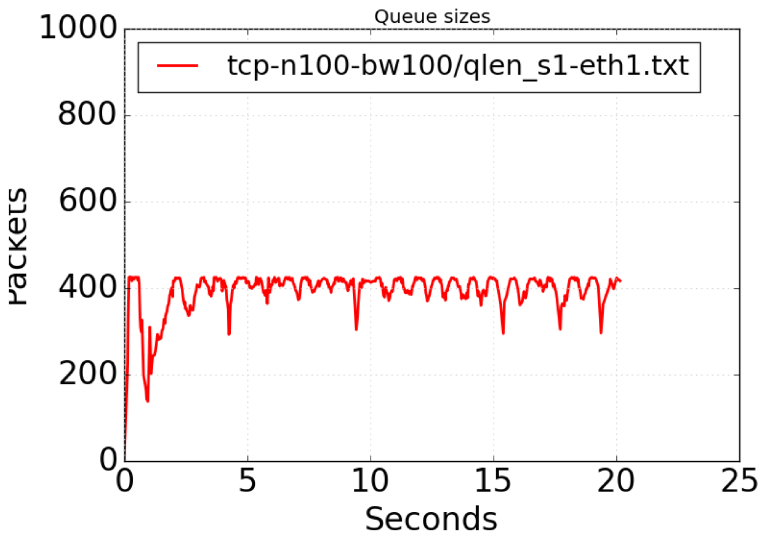

Figure 31: Queue occupancy with bandwidth of 100MBps and 100 hosts for TCP traffic

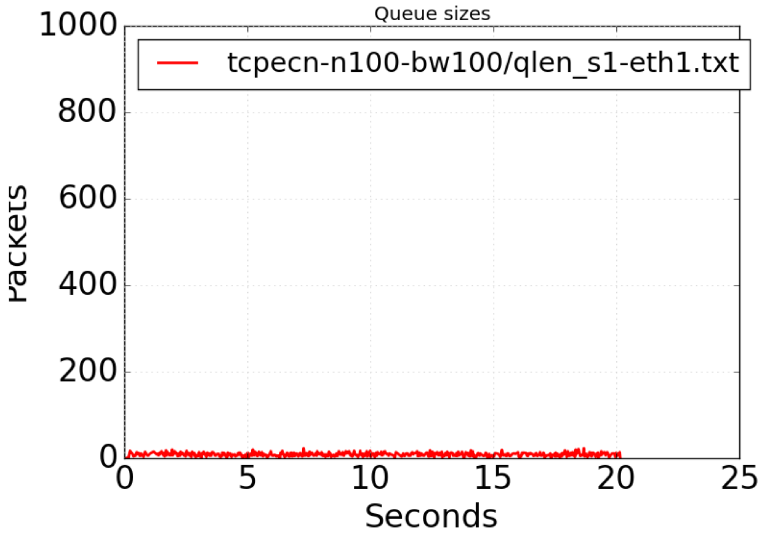

Figure 32: Queue occupancy with bandwidth of 100MBps and 100 hosts for ECN TCP traffic

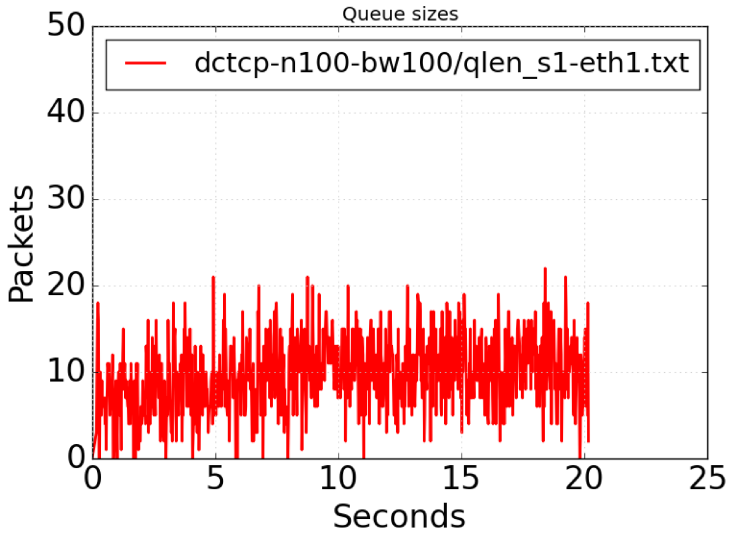

Figure 33: Queue occupancy with bandwidth of 100MBps and 100 hosts for DCTCP traffic

As it is obvious from running the different scenarios and their results that are represented with the figures, a clear picture exists for the benefits of using DCTCP in the links. When the queue occupancy is maintained lower, the variations of CWND so we can see how DCTCP works more smoothly than TCP are shown clearly. It is made obvious that even in scenarios with huge traffic when the number of hosts e.g increases to 100, the queue occupancy is still very low with the use of DCTCP

and that helps with keeping the delays low as not many packets are dropped and the reaction is based on feedback.

Further on the topology of Figure 15 with the same parameters was used to have more results. Figures 34, 35 and 36 show the CWND with the use of this topology for TCP, ECN TCP and DCTCP respectively. The difference in these images is that we focus on the flows and do not use the sum of the CWND so we can see in a smaller scale how the individual flows behave. So the figures are from the hosts h1 and h2 that function as receivers and the flows that reach there, after travelling the bottleneck created between switches s1 and s2. We can make the same observations in a more clear way since the images are more specified.

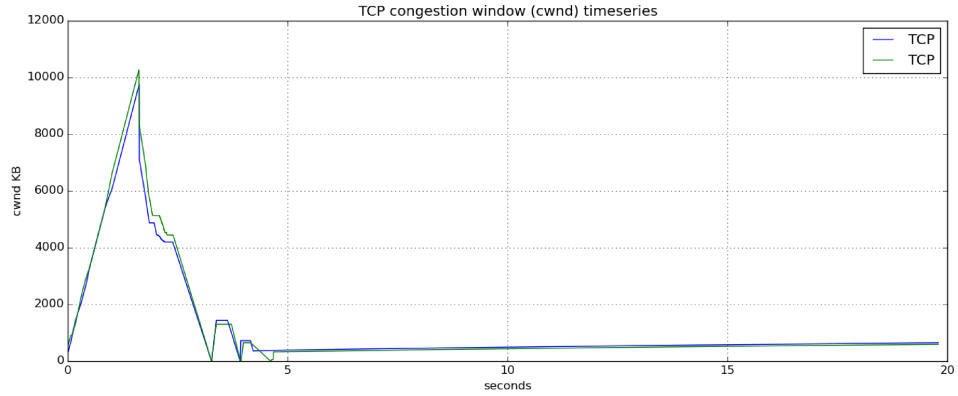

Figure 34: CWND with bandwidth of 100MBps and 4 hosts for the topology of Figure 15 and TCP traffic

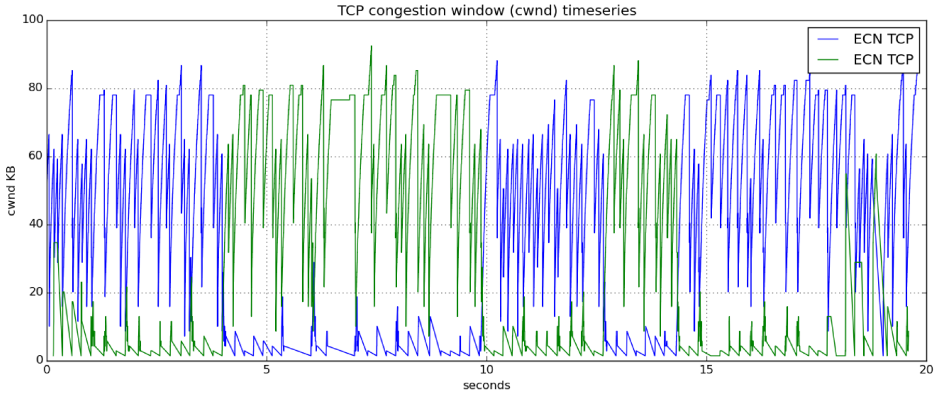

Figure 35: CWND with bandwidth of 100MBps and 4 hosts for the topology of Figure 15 and ECN TCP traffic

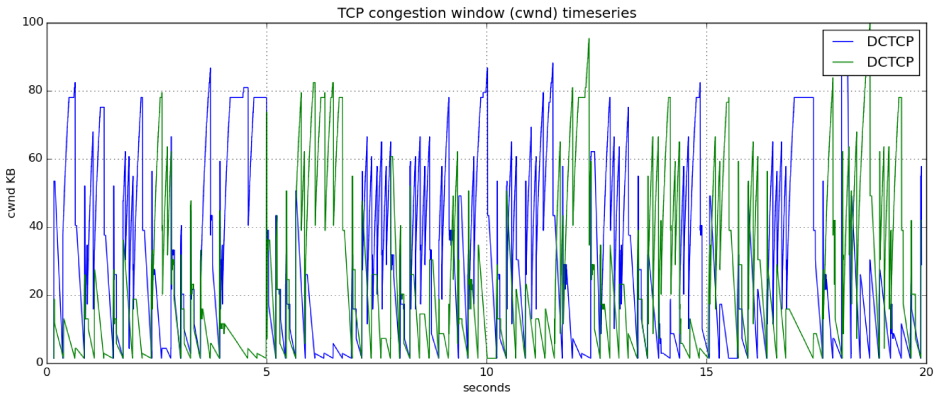

Figure 36: CWND with bandwidth of 100MBps and 4 hosts for the topology of Figure 15 and DCTCP traffic

The initial goal with the use of the topology of Figure 15 was to be able to send simultaneously different kind of traffic from the hosts h3 and h4 so we can have an image of how this traffic traversed the network when they reach hosts h1 and h2. From an example of trying to run simultaneously TCP and DCTCP Figure 37 is the result. The image though was not what we expected, which made us try to understand what was the issue and why the emulator was not functioning as we expected. The solution became apparent when we checked the packets and we understood that because of the limitations of Mininet and the way it uses the kernel, with the change of the congestion control to DCTCP, all the trafficis DCTCP traffic and there is no TCP traffic at all. At the same time the change is not done since the beginning, since Mininet runs TCP for most of the time, until it changes the kernel to DCTCP congestion control and then we see an image that is a mix of the DCTCP and TCP without actually making a lot of sense graphically.

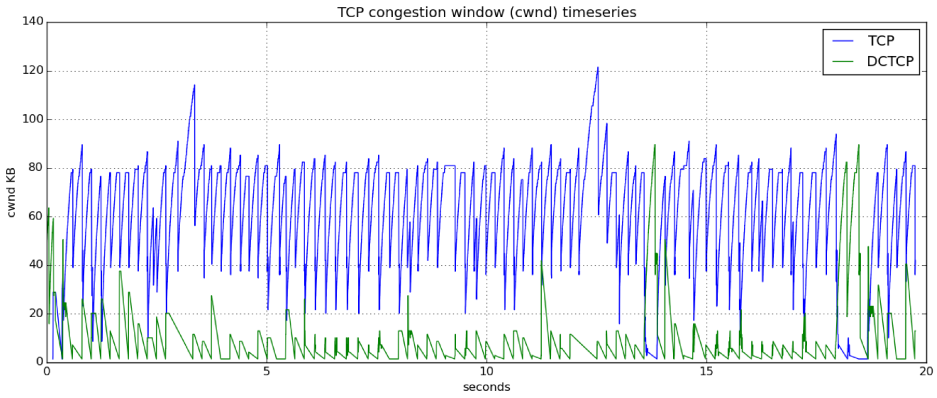

Figure 37: CWND with bandwidth of 100MBps and 4 hosts for the topology of Figure 15 and TCP and DCTCP traffic. It was proved that only DCTCP traffic traverses the network

We could not run simultaneously TCP and DCTCP in the same Mininet environ-

ment but still we want to have an image of the main difference that DCTCP makes on the queue occupancy. For this reason we represented the queue occupancies from a TCP and DCTCP run in the topology of Figure 14 in one graph so we can better see the difference. Figure 38 shows exactly that.

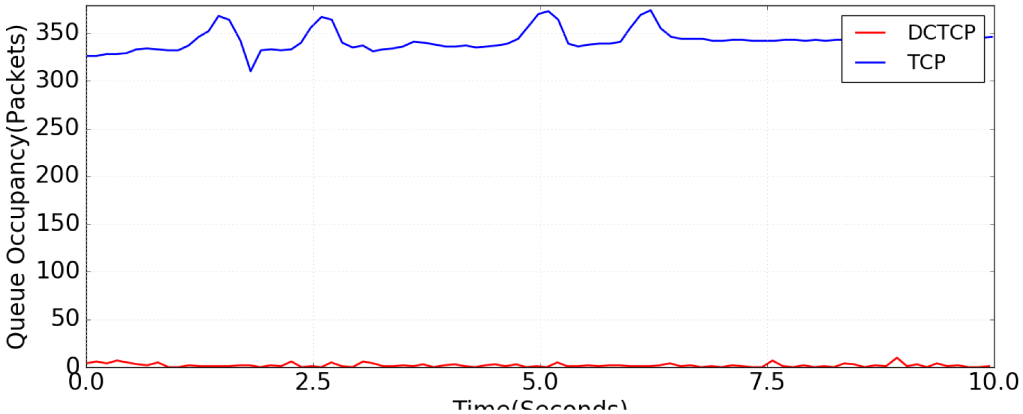

Figure 38: Difference in queue occupancy from TCP and DCTCP traffic

These results have been studied in [1] extensively as well as being simulated and emulated. At the same time, the main reason that DCTCP cannot be deployed in the same links as TCP is because it will lead the links to starvation.

At the same time, to have an idea about the behavior of the protocol in long distance scenarios and comparatively to TCP, we run the protocol with delay of 100ms, that is a scenario of long distance and the extremely long delay of 200ms.

In Figures 41 and 42 we can see the queue occupancy for TCP and DCTCP with 100 ms delay. Again the queue occupancy is very low for DCTCP. At the same time Figure 40, that shows the CWND for DCTCP gives us the image that even though the delay is bigger, DCTCP CWND maintains its lower values comparatively to those of TCP that we can see in Figure 39. Even thought the two CWND look like they have similar behavior from one point onward, DCTCP gains again a lot when it comes to the beginning of the connection, which is much smoother.

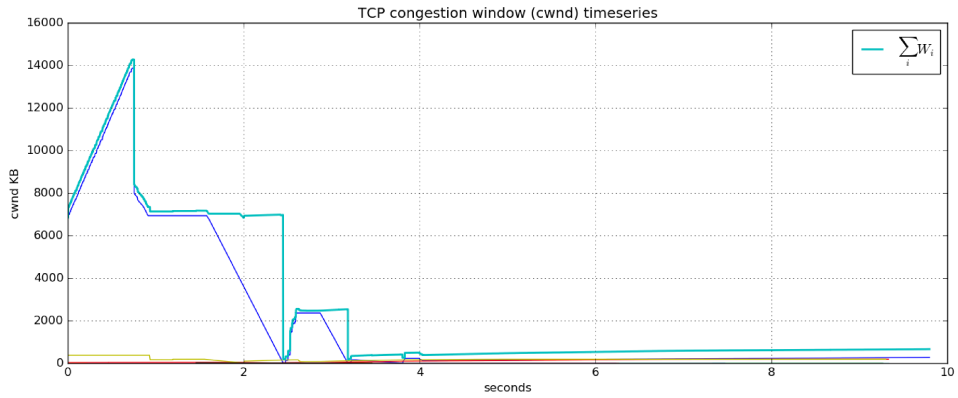

Figure 39: CWND with bandwidth of 100MBps and 100 hosts for the topology of Figure 14 and TCP traffic, 100ms delay

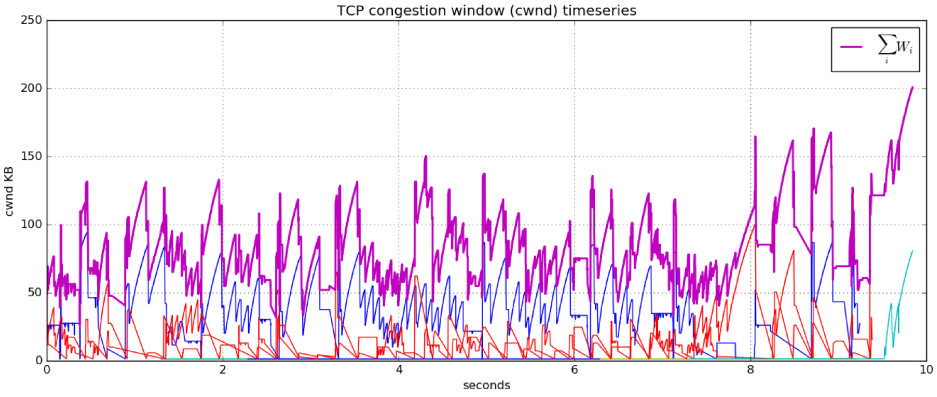

Figure 40: CWND with bandwidth of 100MBps and 100 hosts for the topology of Figure 14 and DCTCP traffic, 100ms delay

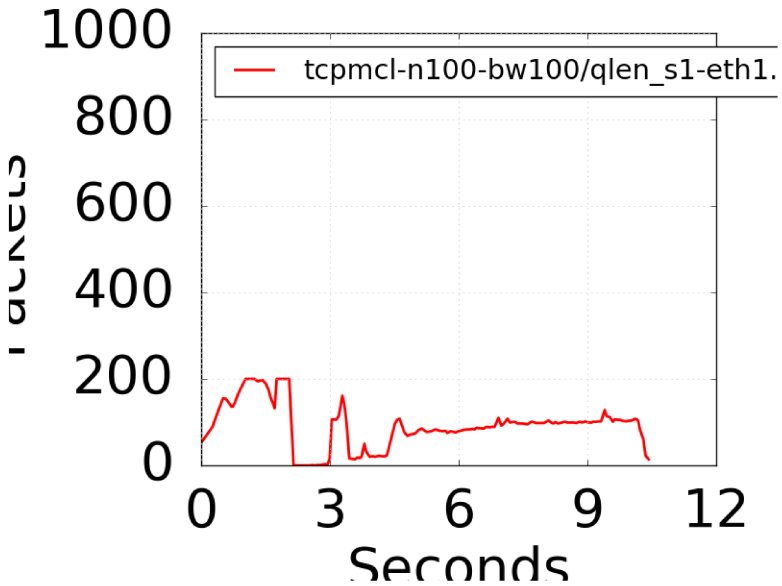

Figure 41: Queue occupancy with bandwidth of 100MBps and 100 hosts for TCP traffic, delay 100ms

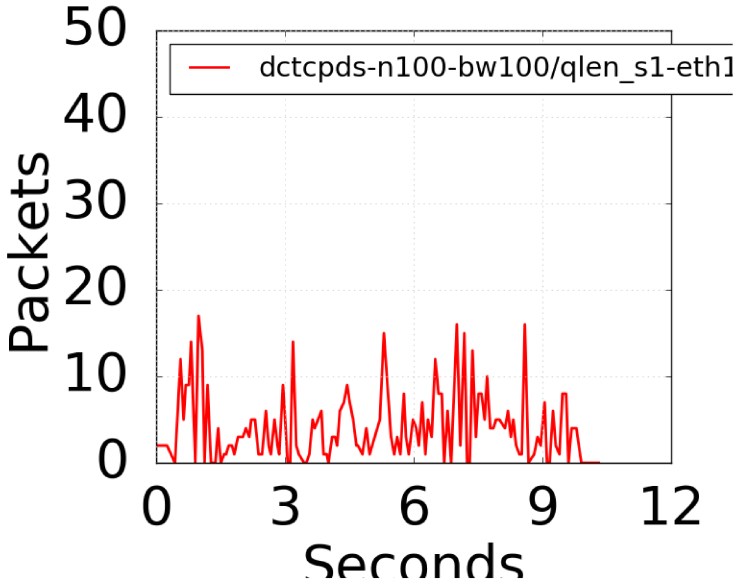

Figure 42: Queue occupancy with bandwidth of 100MBps and 100 hosts for DCTCP traffic, delay 100ms

Similarly, we run the emulator with delay of value 200ms and again we can see the same results. Figure 44 comparatively to Figure 43 shows clearly the smoother behavior of DCTCP when it comes to congestion even in this extreme delay scenario. The queue occupancy is similarly kept lower throughout the transfer as we can see from Figure 46 comparatively to Figure 45 that is the queue occupancy of TCP with delay of 200ms magnitude.

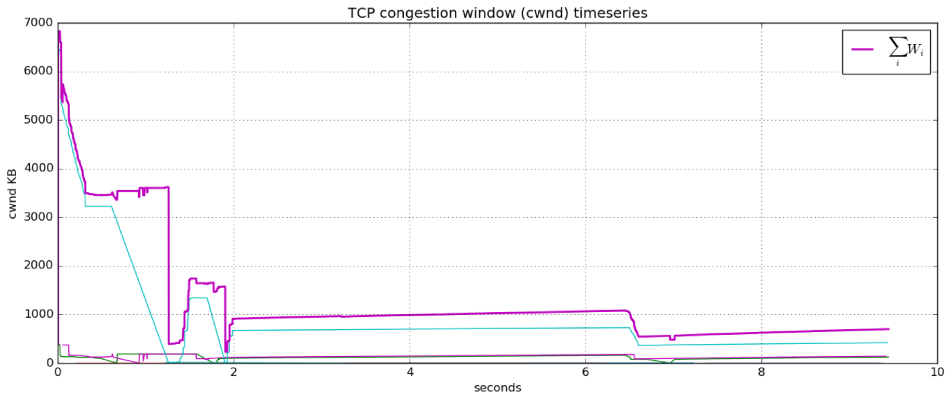

Figure 43: CWND with bandwidth of 100MBps and 100 hosts for the topology of Figure 14 and TCP traffic, 200ms delay

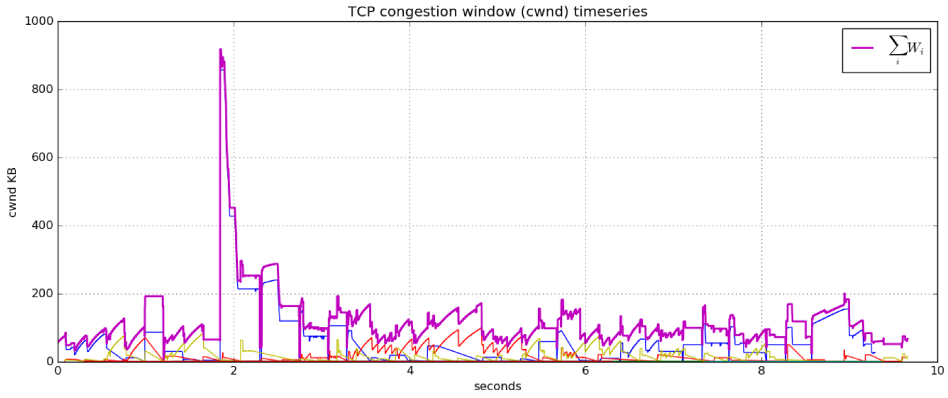

Figure 44: CWND with bandwidth of 100MBps and 100 hosts for the topology of Figure 14 and DCTCP traffic, 200ms delay

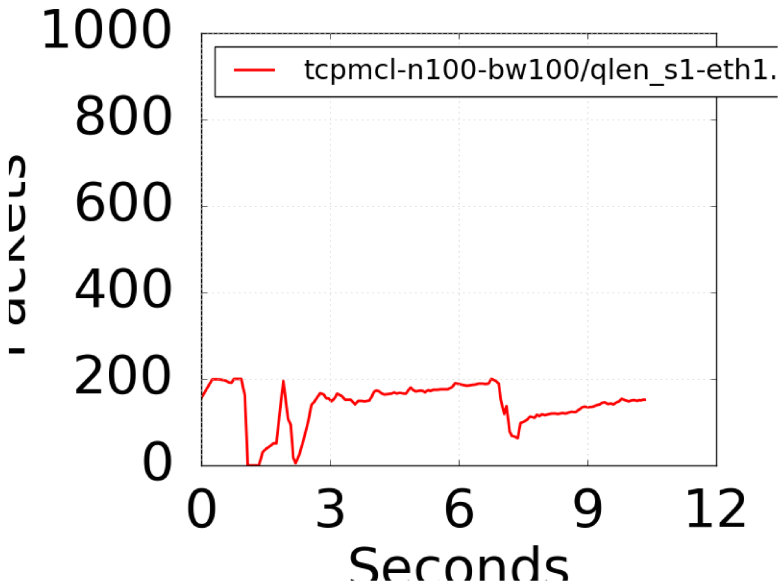

Figure 45: Queue occupancy with bandwidth of 100MBps and 100 hosts for TCP traffic, delay 200ms

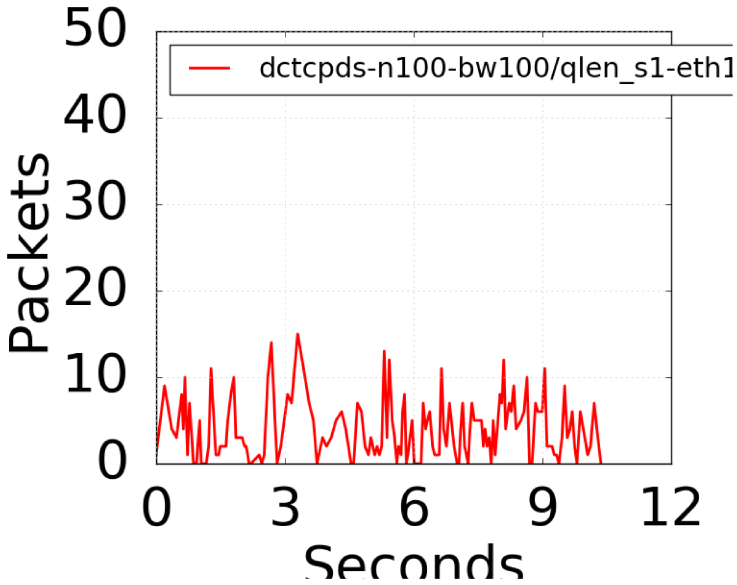

Figure 46: Queue occupancy with bandwidth of 100MBps and 100 hosts for DCTCP traffic, delay 200ms

From our own simulations we managed to prove the benefits of DCTCP use and see how DCTCP behaves in different links with a number of changing parameters. From the study of different scenarios and research it has been further established that there is a growing focus on the deployment of DCTCP in the wide Internet. This happens because of the easy way to change to the protocol with minimal changes and the way DCTCP functions and the results it achieves. Different scenarios can be studied for its Internet deployment, but the main focus again in research is in the

use of ECN feedback, because as it is also shown in simulations, with parametrizing properly, DCTCP flows do not starve the coexisting traffic. This chapter helped the evaluation of the research and draw the conclusions in the next chapter.

7 Conclusion

In the world of today because of the way cloud computing has evolved and the extensive use of data centers, it is extremely important to know how the data center environment functions and what challenges it imposes. One of the main reasons for these challenges to be tackled is the reduction of the expenses to maintain the data center environment.

While there has been extensive study of this problem and a lot of solutions can be proposed, varying from the data center architecture to the way the hardware functions, it is common belief that changes in the protocols used in the data center environment are beneficial as solutions to this problem. In light of this and research showing that most of the traffic that enters data centers is TCP based, a number of studies have been concluded for different variations of TCP and its alternatives especially designed for the data center environment.

The most popular of this alternative data center transport protocols is DCTCP, a variation of TCP that changes the congestion window proportionally to the amount of congestion in the link and based on ECN feedback from middleboxes. At the same time, because it would be beneficial to have homogeneous traffic, it is necessary to study the behaviour of both protocols coexisting in the same environment. To this end, it is necessary to see the behaviour of DCTCP in links that simulate the behaviour of the wide Internet and its behaviour through the coexistence with TCP traffic in the same link.

The initial goal of this thesis was to study the traffic of DCTCP through Mininet inside the data center environment and how the protocol behaves outside inter-data center scenarios. Another aspect that this thesis wanted to cover was the interaction in the links between TCP and DCTCP when they coexist. On the first part the goal was partially achieved, since DCTCP was put across TCP with the use of different delays.

On the second part of what this thesis wanted to achieve the thesis remains inconclusive. Because of the decision to use the Mininet network emulator and the way Mininet functions, in the scope of the thesis we were not able to simultaneously run TCP and DCTCP traffic. For this purpose the better solution would be in future work to use another testbed that is not based on the configurations of the kernel directly. If Mininet is still used for this purpose our conclusion is that it needs to be installed in a separate Virtual machine and the incoming traffic should already be coming from separate machines that have TCP or DCTCP traffic already configured.In addition, a queue that handles differently DCTCP and TCP traffic before passing through the link should be created in the queue that Mininet emulator would have to monitor.

In conclusion to the work that is done by this thesis and also taking into account the research that exists in this topic, DCTCP can be used outside the data center environment. The protocol behaves better than TCP and keeping queue occupancies low even in long propagation delay scenarios, it keeps the losses lower and smoother CWND. At the same time it is not as easy to configure the routers and middle boxes in more complex scenarios, so that would actually impede the smooth use of DCTCP

in broader networks. But if the feedback is accurate, DCTCP gives a low latency, low queue occupancy solution in comparison with the established TCP.

As a result of all the aforementioned, it is proposed that DCTCP should be further studied for wide Internet deployment and especially put in the same links with traffic from other protocols to see its behaviour. At the same time, further research could be done also on the Mininet emulator and how to make it more profitable for these kind of scenarios.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. "Data Center TCP (DCTCP)". *Proc. ACM SIGCOMM 2010 Conference*. SIGCOMM '10. ACM, 2010, pp. 63–74. DOI: 10.1145/1851182.1851192.

[2] M. Alizadeh, A. Javanmard, and B. Prabhakar. "Analysis of DCTCP: Stability, Convergence, and Fairness". *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '11. ACM, 2011, pp. 73–84. DOI: 10.1145/1993744.1993753.

[3] 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". *Proc. 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USENIX Association, 2012, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=2228298.2228324.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. "pFabric: Minimal Near-optimal Datacenter Transport". *Proc. ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. ACM, 2013, pp. 435–446. DOI: 10.1145/2486001.2486031.

[5] F. Baker and G. Fairhurst. *IETF Recommendations Regarding Active Queue Management*. RFC 7567 (Best Current Practice). Internet Engineering Task Force, July 2015. URL: http://www.ietf.org/rfc/rfc7567.txt.

[6] S. Bensley, L. Eggert, D. Thaler, P. Balasubramanian, and G. Judd. *Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters*. Internet-Draft draft-ietf-tcpm-dctcp-01. http://www.ietf.org/internet-drafts/draft-ietf-tcpm-dctcp-01.txt. IETF Secretariat, Nov. 2015. URL: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-dctcp-01.txt.

[7] T. Benson, A. Akella, and D. A. Maltz. "Network Traffic Characteristics of Data Centers in the Wild". *Proc. 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. ACM, 2010, pp. 267–280. DOI: 10.1145/1879141.1879175.

[8] T. Benson, A. Anand, A. Akella, and M. Zhang. "Understanding Data Center Traffic Characteristics". *SIGCOMM Comput. Commun. Rev.* 40.1 (Jan. 2010), pp. 92–99. DOI: 10.1145/1672308.1672325.

[9] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. *Recommendations on Queue Management and Congestion Avoidance in the Internet*. RFC 2309 (Informational). Obsoleted by RFC 7567, updated by RFC 7141. Internet Engineering Task Force, Apr. 1998. URL: http://www.ietf.org/rfc/rfc2309.txt.

[10] W. Chen, P. Cheng, F. Ren, R. Shu, and C. Lin. "Ease the Queue Oscillation: Analysis and Enhancement of DCTCP". *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. July 2013, pp. 450–459. DOI: 10.1109/ICDCS.2013.22.

[11] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. "Understanding TCP Incast Throughput Collapse in Datacenter Networks". *Proc. 1st ACM Workshop on Research on Enterprise Networking*. WREN '09. ACM, 2009, pp. 73–82. DOI: 10.1145/1592681.1592693.

[12] R. D. S. Couto, S. Secci, M. E. M. Campista, and L. H. M. K. Costa. "Reliability and Survivability Analysis of Data Center Network Topologies". *CoRR* abs/1510.02735 (2015). URL: http://arxiv.org/abs/1510.02735.

[13] N. Dukkipati. "Rate Control Protocol (Rcp): Congestion Control to Make Flows Complete Quickly". AAI3292347. PhD thesis. 2008.

[14] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. "Hedera: Dynamic Flow Scheduling for Data Center Networks". *Proc. 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. USENIX Association, 2010, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=1855711.1855730.

[15] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". *IEEE/ACM Trans. Netw.* 1.4 (Aug. 1993), pp. 397–413. DOI: 10.1109/90.251892.

[16] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. "Reproducible Network Experiments Using Container-based Emulation". *Proc. 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '12. ACM, 2012, pp. 253–264. DOI: 10.1145/2413176.2413206.

[17] C.-Y. Hong, M. Caesar, and P. B. Godfrey. "Finishing Flows Quickly with Preemptive Scheduling". *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. ACM, 2012, pp. 127–138. DOI: 10.1145/2342356.2342389.

[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. "The Nature of Data Center Traffic: Measurements & Analysis". *Proc. 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '09. ACM, 2009, pp. 202–208. DOI: 10.1145/1644893.1644918.

[19] M. Kuhlewind, D. Wagner, J. Espinosa, and B. Briscoe. "Using data center TCP (DCTCP) in the Internet". *Globecom Workshops (GC Wkshps), 2014*. Dec. 2014, pp. 583–588. DOI: 10.1109/GLOCOMW.2014.7063495.

[20] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach (4th Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2007.

[21] A. Leon-Garcia and I. Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures*. 1st. McGraw-Hill School Education Group, 1999.

[22] Y. Liu, J. K. Muppala, and M. Veeraraghavan. "A Survey of Data Center Network Architectures". 2013.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks". *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. DOI: 10.1145/1355734.1355746.

[24] P. M. Mell and T. Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep. 2011.

[25] 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". *Proc. 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. ACM, 2014, pp. 491–502. DOI: 10.1145/2619239.2626305.

[26] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896 (Historic). Obsoleted by RFC 7805. Internet Engineering Task Force, Jan. 1984. URL: http://www.ietf.org/rfc/rfc896.txt.

[27] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. *Controlled Delay Active Queue Management*. Internet-Draft draft-ietf-aqm-codel-04. http://www.ietf.org/internet-drafts/draft-ietf-aqm-codel-04.txt. IETF Secretariat, June 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-aqm-codel-04.txt.

[28] 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". *SIGCOMM Comput. Commun. Rev.* 39.4 (Aug. 2009), pp. 39–50. DOI: 10.1145/1594977.1592575.

[29] R. Pan, P. Natarajan, F. Baker, and G. White. *PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem*. Internet-Draft draft-ietf-aqm-pie-08. http://www.ietf.org/internet-drafts/draft-ietf-aqm-pie-08.txt. IETF Secretariat, June 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-aqm-pie-08.txt.

[30] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. "Fastpass: A Centralized "Zero-queue" Datacenter Network". *Proc. 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. ACM, 2014, pp. 307–318. DOI: 10.1145/2619239.2626309.

[31] P. Prakash, A. Dixit, Y. C. Hu, and R. Kompella. "The TCP Outcast Problem: Exposing Unfairness in Data Center Networks". *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX, 2012, pp. 413–426. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/prakash.

[32] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. "Improving Datacenter Performance and Robustness with Multipath TCP". *Proc. ACM SIGCOMM 2011 Conference*. SIGCOMM '11. ACM, 2011, pp. 266–277. DOI: 10.1145/2018436.2018467.

[33] R. P. Tahiliani, M. P. Tahiliani, and K. C. Sekaran. "TCP Variants for Data Center Networks: A Comparative Study". *Cloud and Services Computing (ISCOS), 2012 International Symposium on*. Dec. 2012, pp. 57–62. DOI: 10.1109/ISCOS.2012.38.

[34] B. Vamanan, J. Hasan, and T. Vijaykumar. "Deadline-aware Datacenter TCP (D2TCP)". *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012), pp. 115–126. DOI: 10.1145/2377677.2377709.

[35] 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". *Proc. ACM SIGCOMM 2009 Conference on Data Communication*. SIGCOMM '09. ACM, 2009, pp. 303–314. DOI: 10.1145/1592568.1592604.

[36] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. "Better Never Than Late: Meeting Deadlines in Datacenter Networks". *Proc. ACM SIGCOMM 2011 Conference*. SIGCOMM '11. ACM, 2011, pp. 50–61. DOI: 10.1145/2018436.2018443.

[37] H. Wu, Z. Feng, C. Guo, and Y. Zhang. "ICTCP: Incast Congestion Control for TCP in Data Center Networks". *Proc. 6th International COnference*. Co-NEXT '10. ACM, 2010, 13:1–13:12. DOI: 10.1145/1921168.1921186.

[38] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks". *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012), pp. 139–150. DOI: 10.1145/2377677.2377711.

[39] Y. Zhang and N. Ansari. "On Architecture Design, Congestion Notification, TCP Incast and Power Consumption in Data Centers". *IEEE Communications Surveys Tutorials* 15.1 (First 2013), pp. 39–64. DOI: 10.1109/SURV.2011.122211.00017.