SDN Based Solution for Detecting and Mitigating Incast in Data Centers

By

Muhammad Aqib Sarfraz

00000204860

Supervisor

Asst Prof Yawar Abbas Bangash, PhD

Department of Electrical Engineering

Military College Of Signals (MCS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

July 2020

SDN Based Solution for Detecting and Mitigating Incast in Data Centers

By

Muhammad Aqib Sarfraz

00000204860

Supervisor

Asst Prof Yawar Abbas Bangash, PhD

A thesis submitted in conformity with the requirements for the degree of *Master of Science* in

Electrical Engineering

Department of Electrical Engineering

Military College Of Signals (MCS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

July 2020

Thesis Acceptance Certificate

Certified that final copy of MS thesis written by **Mr. Muhammad Aqib Sarfraz**, Registration No.**00000204860** of Military College of Signals has been vetted by undersigned, found complete in all respect as per NUST Statues/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor: **Asst Prof Yawar Abbas Bangash, PhD**

Date: ____________

Signature (HOD): ____________

Date: ____________

Signature (Dean): ____________

Date: ____________

Declaration

I, *Muhammad Aqib Sarfraz* declare that this thesis titled "SDN Based Solution for Detecting and Mitigating Incast in Data Centers" and the work presented in it are my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Muhammad Aqib Sarfraz,
00000204860

Copyright Notice

Dedication

This thesis is dedicated to *my beloved parents, teachers and siblings*

Abstract

The Internet of Things (IoT) and the Cloud Computing are the two emerging technologies which have seen a wide-scale adoption. Many IoT applications require cloud platform support for computing and storage purpose. Hence managing network resources and congestion control in a cloud data center network (DCN) environment have become a major concern. In a large scale DCN, there are several storage and network devices interconnected. Achieving high performance in an environment having low latency, high throughput and shallow buffered network devices, poses a great challenge for network administrators. In such situation many-to-one TCP communication pattern predominates, which often results in an individual link throughput collapse known as Incast. In this paper, we present a centralized approach, called as modified dynamic fair-share buffer policy (M-DyFaShBuP), which operates in a high bursty traffic environment. The software-defined network (SDN) controller plays an imperative role from policy designing till policy implementation. An equitable share for each server is calculated and distributed by the controller among all storage servers. The storage servers adjust their sending rates, according to the received policy. We incorporate top-of-rack (TOR) and end-of-row (EOR) designs while handling Incast. M-DyFaShBuP successfully tackles the Incast scenario and achieves high throughput performance.

Keywords: *Cloud Computing, DCN, EOR, fair-share, Incast, IoT, SDN, TCP, TOR*

Acknowledgments

First I am thankful to ALMIGHTY ALLAH who gave me strength and wisdom to complete this work. Moreover, I am also thankful to my supervisor and committee members for their continuous guidance, support and encouragement throughout this research work. In the end I am grateful to my parents for their support and prayers.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

ACK	Acknowledgement
DC	Data Center
DCN	Data Center Network
EOR	End-of-row
IoT	Internet of things
SDN	Software-defined networking
TCP	Transport control protocol
TOR	Top-of-rack

CHAPTER 1

Introduction

The term Internet of Things (IoT) was coined in 1999 by Kevin Ashton [1]. It aims to allow everything around us to have a universal connectivity [2]. It is a kind of network in which various objects, devices, vehicles, buildings, etc. are embedded with sensors, actuators and software, are connected with internet and sharing data [3]. The Cloud Computing is a kind of an infrastructure which gives on-demand provision of various resources such as servers, storage, networks, applications and services among different users [4]. Many IoT devices sense or collect data and send it to the Cloud for further processing or storage purpose [5]. In IoT, the data center (DC) is the key infrastructure for those applications which require massive amount of data storage and data processing [6].

1.1 Data Center Network

DC provide many distributed services such as: Storage of data in bulk and processing of enormous amount of data. These data centers are required to host a variety of web applications and to store a massive amount of data [7]. They are now considered as an important networking infrastructure to run the internet applications and services. Inside a DC there are large number of storage and computational devices which are interconnected through a special network called as data center network (DCN) [8]. These networks are usually designed to attain low latency and high throughput with different load conditions [9]. As the DCNs are rapidly growing in size, so as the number of servers are also increasing exponentially. The number of servers in a Microsoft data center becomes double after every 14 months because of high demand of processing and storage [10]. High quality services of DC are directly linked with the design of DCN. Hence the DCN must be optimized to achieve high efficiency and low cost. It can be classified into two sub types based on the type of networking equipment used: DCN with distributed architecture and DCN with centralized architecture.

1.1.1 Data Center Network with Distributed Architecture

The transport protocols running inside switches and routers, and the distributed control are the key technologies which are responsible for transmitting information in traditional IP networks. Similarly, traditional DCNs are designed in such a way that the storage and computational devices are widely spread across thousands of servers in a distributed way [11]. The switches which are being used in traditional DCN have data plane and control plane vertically integrated as shown in Figure 1.1. The control plane make decisions for traffic flow and the data plane forwards the traffic in line with the decisions made by the control plane.

In these networks each individual network device have its own control plane, due to which the network operator have to configure each network device separately [12]. The network functionality resides inside a networking device in case on traditional networks as shown in Figure 1.2. The distributed control is the reason due to which these networks are tiresome to manage and are complicated in configuration. Also traditional DCNs are unable to support large amount of data [13].

1.1.2 Data Center Network with Centralized Architecture

Software-defined networking (SDN) has given a hope in networking industry, particularly in DCN to solve the traditional DCN challenges [12]. SDN has gained tremendous success commercially such as Google have deployed software-defined network globally to interconnect its data centers. SDN has improved the operational efficiency and sig-

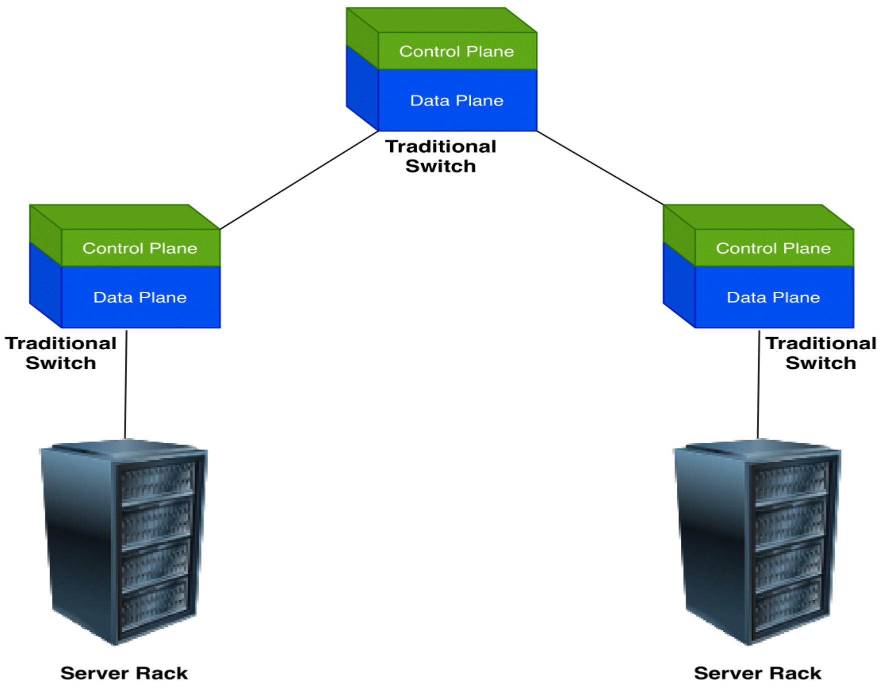


Figure 1.1: Traditional DCN: Data plane and control plane both resides in switch.

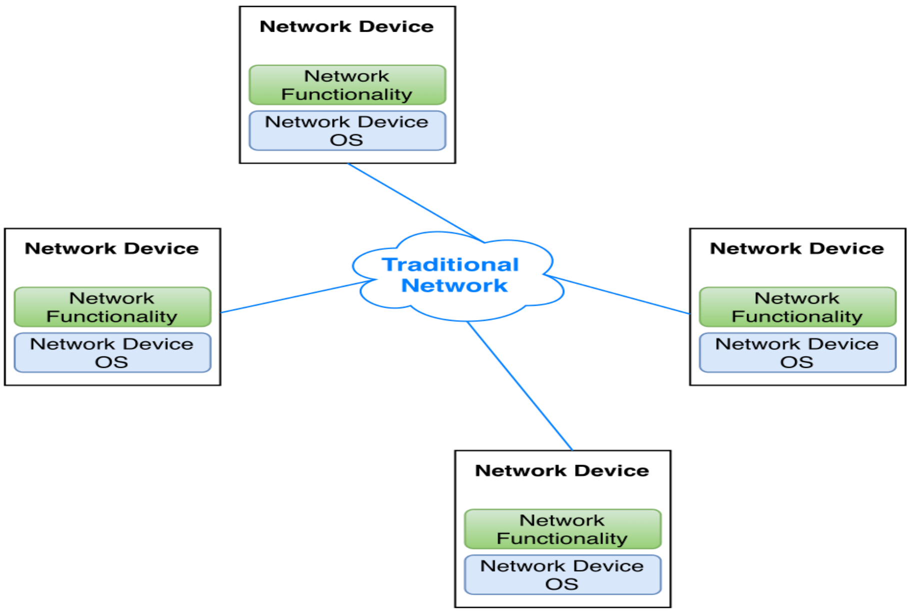


Figure 1.2: Traditional Networking Device: Switch makes the networking decisions.

nificantly reduced the running cost of the company [14]. SDN is prominent due to its eminent features of separation of data plane and control plane. SDN based DCN have a logically centralized controller as shown in Figure 1.3 and Figure 1.4 shows that the network functionality resides inside the SDN Controller [15]. Due to which it provides global network overview, efficient configuration, better performance and higher flexibility [8].

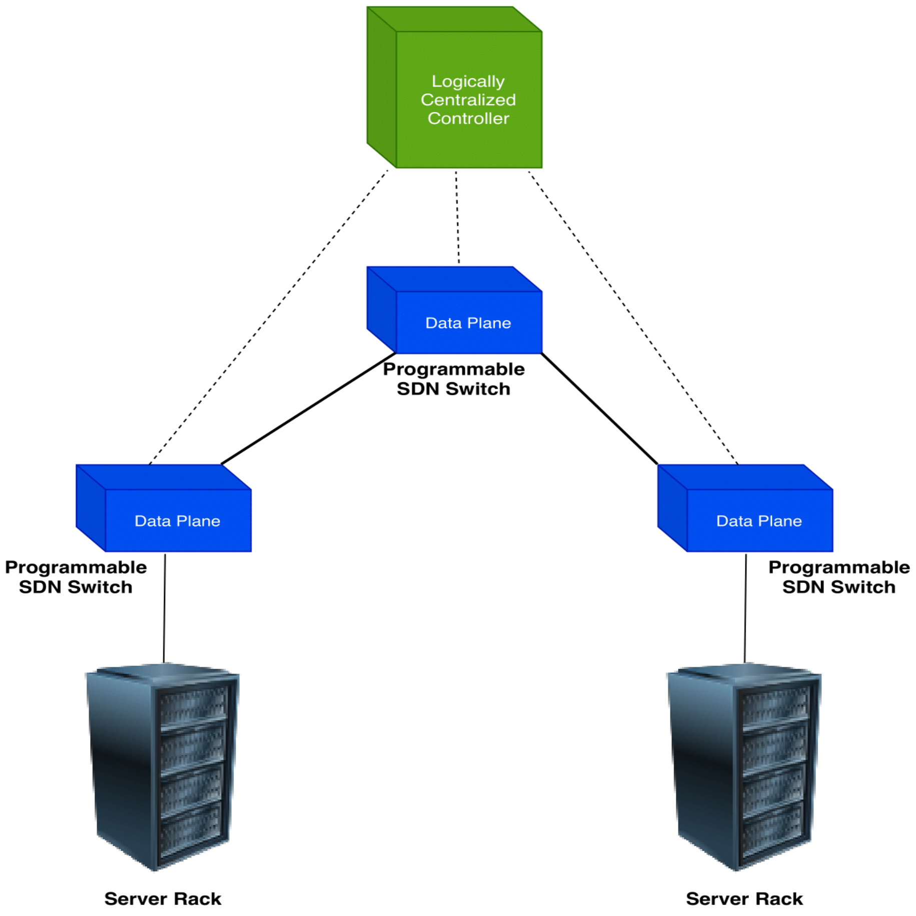


Figure 1.3: SDN based DCN: Data plane resides in switch while control plane is part of a centralized controller.

1.2 Performance Issues in Data Centers

In data centers the links with core switch are heavily utilized and the links with the edge switches face more losses [16]. The traffic inside data center has some unique characteristics due to which it faces some performance issues such as Incast, flow com-

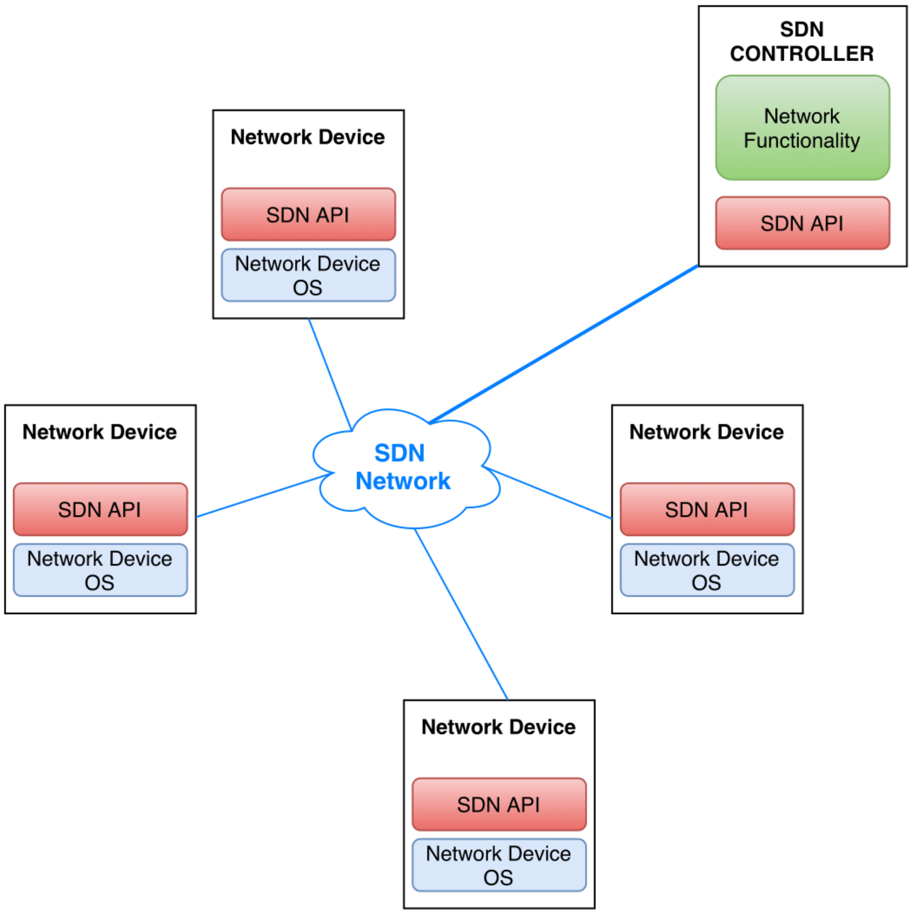


Figure 1.4: SDN based Networking Device: Controller makes the networking decisions.

pletion time etc. Our work is focused on finding an appropriate SDN based solution for avoiding Incast in DC. Incast results in increase in latency which is a very serious issue for modern data centers. It was observed earlier that an increase of 100ms delay would results in 1 percent loss in Amazon sales. Also Google losses its 20 percent traffic for every 500 ms delay while responding a search query [17, 18]. Therefore, an efficient response time is required from a DCNs in a low latency environment [19, 20].

1.2.1 TCP Incast

DCN as compared to IP Networks have some distinguished features like many to one communication pattern followed by low latency, shallow buffered switches and multi rooted tree topology [21, 22]. Transport Control Protocol (TCP) contributes to the maximum traffic inside a DC and TCP is also considered as a backbone of the modern internet [23]. As per the distinguished features of the DCN, when TCP protocol is deployed in DCN, it requires to meet some special properties of DCN such as: Rare packet loss events and fast convergence. However, conventional TCP does not meet these requirements of the DCN [24]. So, if the conventional TCP is deployed in DCN, the traffic burst generated in DCs produces enormous amount of congestion on the bottleneck switch. Incast is a phenomena, as shown in Figure 1.5, which arises in DC when many storage servers send parallel requests to a single receiving server [25]. The small buffers of the bottleneck switch gets exhaust; hence, causing massive amount of packet drops and bursty retransmissions.

During Incast when a packet loss event occurs, TCP can face a timeout that can last a

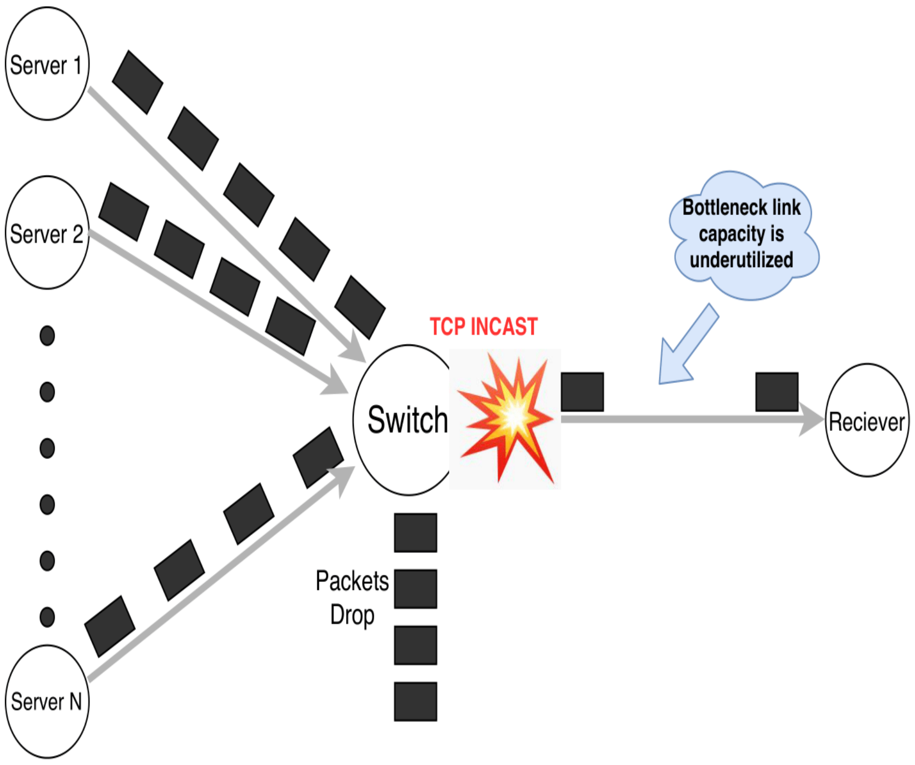


Figure 1.5: TCP Incast scenario: Many servers send data simultaneously to a single receiver causing switch buffer to overflow.

minimum of 200ms [26]. If any server experiences a timeout event, its response is delayed by a minimum of 200ms; meanwhile other servers can complete their responses, but the client must wait until the delayed response is completed. During this delayed response period the client's link may get idle or the client's link throughput is drastically decreased, this phenomena is commonly known as TCP throughput collapse [27, 28], as shown in Figure 1.6. It can be seen, when the number of servers exceeds 7, the client's link throughput is drastically decreased [29]. An overview of Incast can be found in [30].

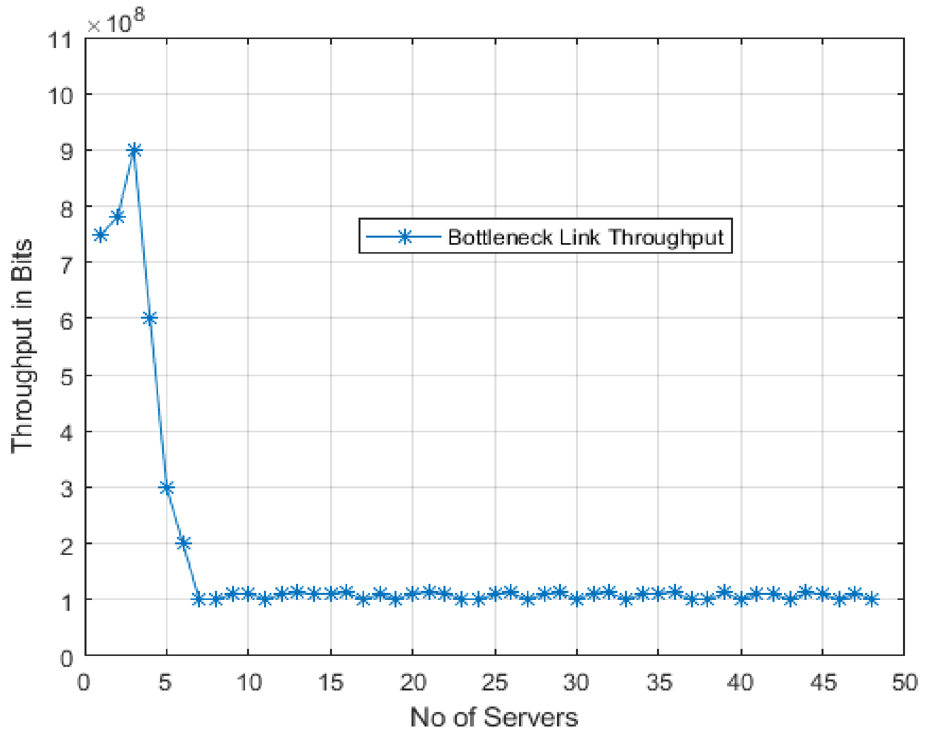


Figure 1.6: TCP Throughput Collapse: As the number of servers increase the TCP throughput drops significantly [28].

1.3 Proposed Solution

Many techniques have been presented to solve the TCP Incast problem, but their target was either to reduce the TCP retransmission timeout [21], or to manage the switch buffer occupancy in order to restrict the buffer overflow by utilizing explicit congestion notification (ECN) and modifying the TCP on both the sender and receiver ends [31]. Also a few of the existing methods are good enough to handle congestion but still they lag the ability to fully utilize the bottleneck link capacity.

A software-defined networking **(SDN)** based approach is required which can easily tackle the Incast. We have proposed a few modifications in dynamic fair-share buffer policy (DyFaShBuP) [32], and named it as modified dynamic fair-share buffer policy **(M-DyFaShBuP)**. A broader perspective about working of M-DyFaShBuP is shown in Figure 1.7. In this figure, different steps are shown which are carried out in our proposed method. The controller is responsible for making a policy and distributing it among the storage nodes. In our proposed method, our **objective** is to provide an SDN based solution which encompasses;

1. An uncomplicated process to tackle Incast congestion.
2. Elimination of the conventional ECN based detection method.
3. Fair distribution of the policy among all servers.
4. Complete utilization of the bottleneck link capacity.

We have increased the scope DyFaShBuP, by implementing it on both top-of-rack (TOR) switches and end-of-row (EOR) switches. We have introduced a slight change in method for dynamically calculating the number of servers. Our experimental results show that M-DyFaShBuP easily handles the Incast issue and it also fully utilizes the bottleneck link capacity even if the large number of servers are sending data.

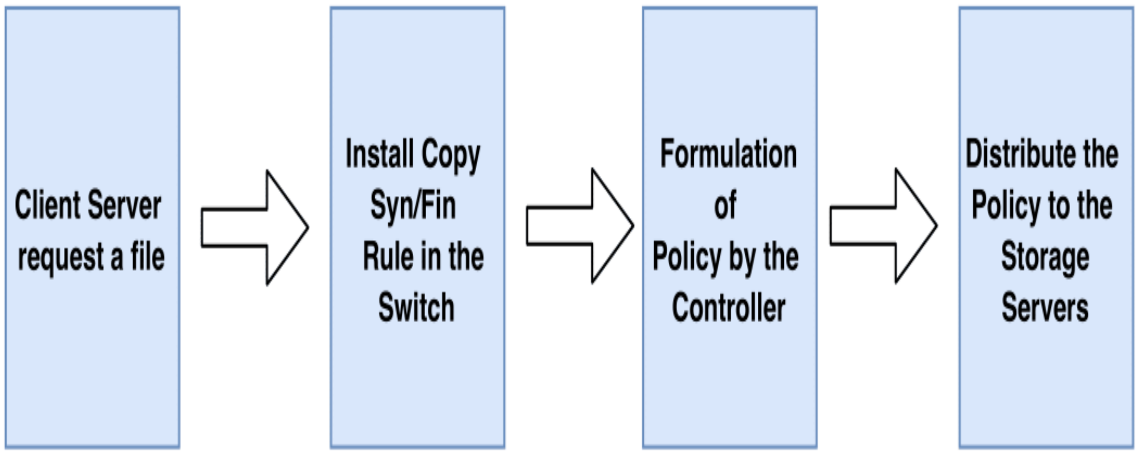

Client Server request a file
Install Copy Syn/Fin Rule in the Switch
Formulation of Policy by the Controller
Distribute the Policy to the Storage Servers

Figure 1.7: General idea about M-DyFaShBup: The process starts as the client request a file which is stored on multiple storage servers.

1.4 Thesis Organization

Chapter 2 discusses IoT and SDN based cloud data centers. Chapter 3 discusses previous work regarding Incast. Chapter 4 describes the architecture and proposed method. Chapter 5 discusses the results and analysis. Chapter 6 concludes the thesis document.

CHAPTER 2

IoT and SDN Based Cloud Data Centers

2.1 Internet of Things

The IoT is a kind of a concept that every device on the globe is connected with the internet. The IoT devices have actuators and sensors. These devices collect measurements by the help of sensors and send them to a relevant IoT application, that is deployed in the cloud data center. These devices repeatedly collect and forward the data for analysing and processing on a large scale. This data is need to be stored and utilized intelligently for actuation and smart monitoring [33]. It is expected that everything around us will be connected to the internet because of the ongoing trend of ubiquitous cloud computing [34]. The IoT is transforming the process through which day-to-day tasks are carry out. Considering the automated home as an example: The electronic devices can be monitored and controlled remotely by using a smart phone. The IoT is also utilized as a tool in business environment. It covers variety of applications like transport, utility, healthcare etc. [35].
The IoT devices generate Big Data [36], which impose a lot of load on the internet infrastructure. Hence, most of the organizations go for cloud data centers to decrease the burden caused by this load. Rather making a computing and a storage infrastructure on premises, the Cloud Computing allow companies to utilize Virtual Machines (VM). Ta-

Table 2.1: Relationship between IoT and Cloud Computing

Technology	**Big Data**	**Storage Space**	**Computing Capabilities**	**Business Activities**
IoT	It generates Big Data	Small	Limited	It generates business activities
Cloud Computing	It provides the way to handle and store Big Data	Large	Virtually unlimited	It provides the solution to expand business activities

ble 2.1 display the connection between Cloud Computing and IoT. As the total number of IoT devices and objects are increasing rapidly, so as the amount of data generated by them is also growing at an unprecedented rate. It is estimated, that by 2025 the total number of IoT objects will be around 75.44 billion [37–39]. Hence, the dependency of IoT devices on the cloud is also increasing.

2.2 IoT Architecture

The IoT devices and objects are connected with the internet via an access point while the IoT applications are deployed in a data center network. Figure 2.1 shows that the IoT devices are part of access network and the IoT applications are part of a DCN. In our work, we are considering the DCN where IoT applications are deployed. The data which is sent by the IoT devices is processed by the IoT applications. The IoT developers give rent to the cloud solution providers for utilizing the infrastructure on which the IoT application is deployed.

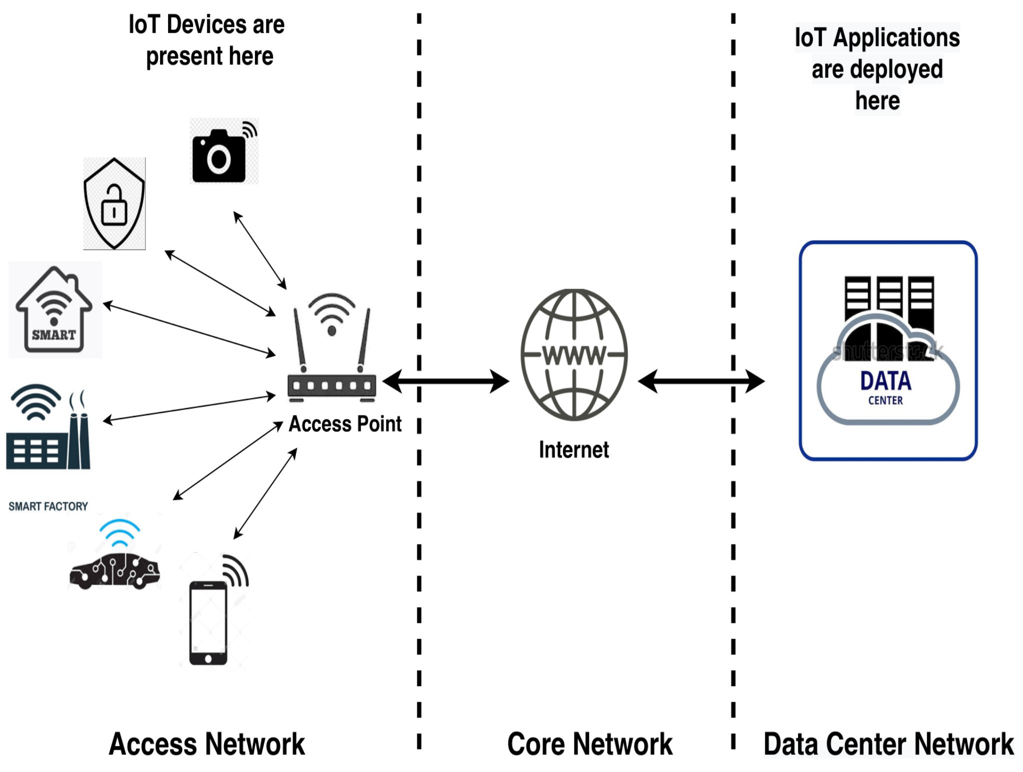


Figure 2.1: A broader perspective of an IoT architecture: IoT devices are part of the access network while IoT applications are part of data center network.

2.3 Issues with Traditional Network Devices

The networking devices in a traditional network are preprogrammed with complicated rules; hence, these rules are difficult to modify in real-time for completing a task. In addition, these traditional devices are resource-constrained, so they lag the capability of installing multiple rules for giving an optimal network service [40]. Moreover, the network administrators also configure protocols manually via low level commands, by using command line interface (CLI) [41]. Hence, these networks are not able to adapt policies that are required to fulfil application specific-requirements in real-time for different IoT applications.

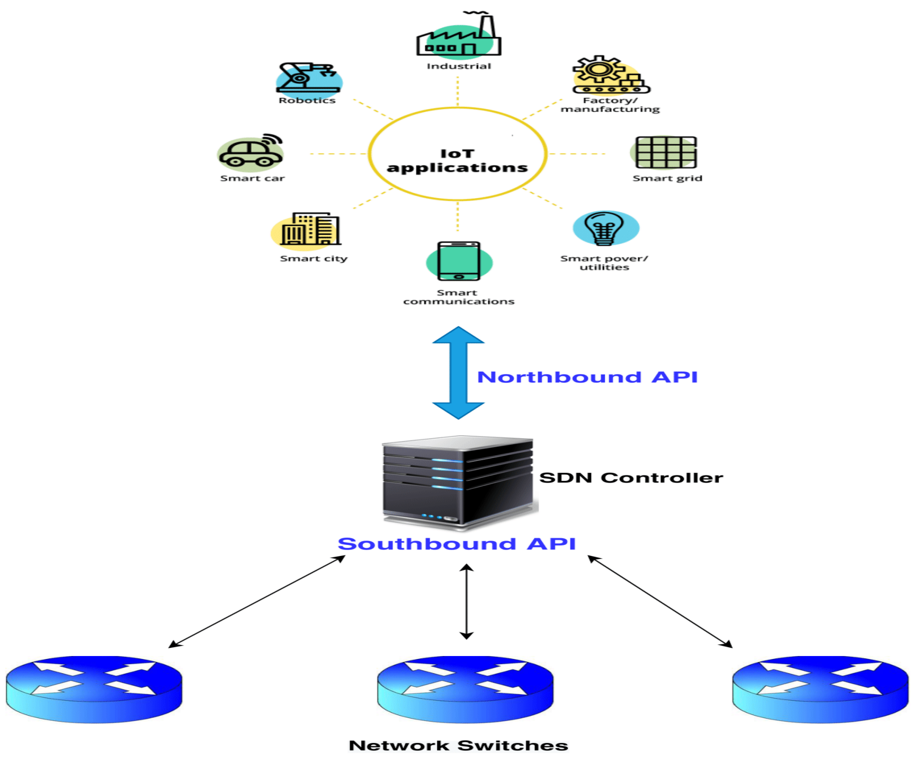


Figure 2.2: An IoT structure based on SDN: IoT applications communicate with the SDN controller via Northbound API and the SDN controller communicates with the network switches via Southbound API.

2.4 SDN Based Cloud Data Centers

The SDN paradigm is more appropriate for handling the network cloud because SDN gives centralized resource management. Consequently, it is more suitable for providing real-time performance in case of IoT based cloud applications [42]. The Northbound application programming interface (API) is utilized for connection between the SDN controller, and the IoT application and Southbound API is utilized for connection between the network switches and the SDN controller as shown in Figure 2.2 [43].

CHAPTER 3

Related Work

The problem of Incast Congestion in a data center was initially identified by Nagle et al. in [29]. Since then the efforts on controlling congestion have gain momentum. These work can be divided into two categories;

1. Tackling Incast via distributed approach.
2. Tackling Incast via centralized approach.

Figure 3.1 shows a few of the techniques which were utilized earlier for handling the Incast problem.

3.1 Tackling Incast via Distributed Approach

In this section we will discuss the traditional TCP Incast mitigation techniques for DCN. These approaches are decentralized.

3.1.1 Fine-grained RTO

In IP networks the default value for the TCP retransmission time out (RTO) is 200msec, while the round trip time (RTT) for DC is 0.1msec. In case of many to one communication pattern inside DCN, if any flow faces RTO it will be delayed by 200msec. This difference of time between RTO and DC RTT causes severe performance degradation

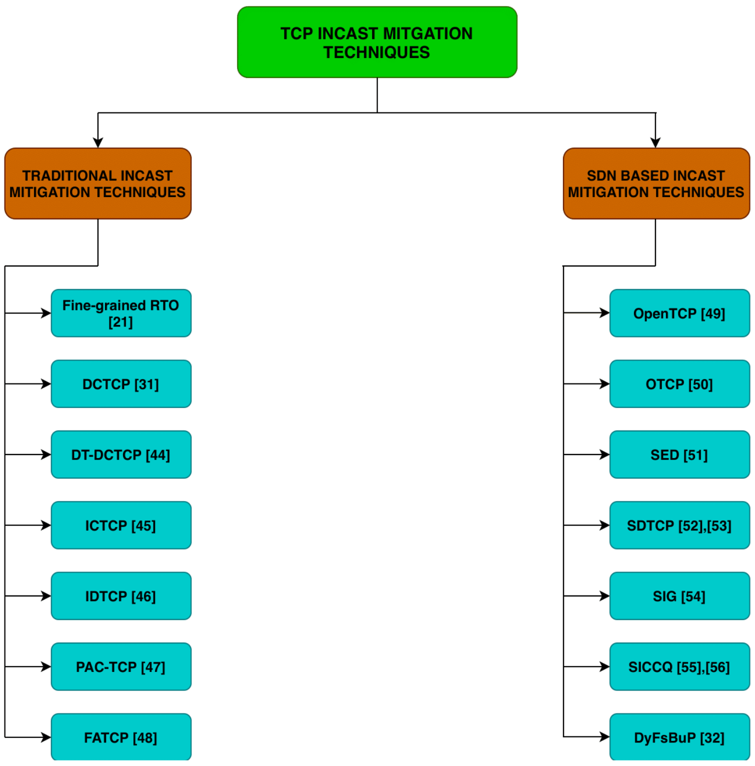


Figure 3.1: Different TCP Incast mitigation methods: Using distributed approach and using centralized approach.

for TCP inside DCN. Fine-grained TCP RTO is one of the initial techniques which was proposed by Vasudevan et al. [21] to tackle incast congestion problem in DCNs. The technique found that by using microsecond-granularity TCP RTO instead of minimum RTO can control Incast. However, the value of fine-grained RTO can effect when servers try to communicate with clients outside the DCN. Most of systems do not have such high-resolution timers for measuring such low RTO values.

3.1.2 DCTCP

Alizadeh et al. [31] presented a protocol named as Data Center TCP (DCTCP) which was designed to operate in a DCN environment. DCTCP algorithm can be dived into three parts;

- Active Que Management (AQM) scheme is used to notify the senders in case of buffer over flow. As soon as the queue occupancy inside switch buffer crosses the threshold ‘K’ the arriving packets are now marked as Congestion Experienced (CE).

- The exact sequence of the marked packed (CE) is conveyed to the senders by sending ACKs.
- The sender reduces the size of congestion window according the status of congestion in the network as perceived by the marked packets.

According to the experimental results, DCTCP could not solve the incast problem when large number of senders are sending parallel requests.

3.1.3 DT-DCTCP

Chen et al. [44] proposed few modifications in DCTCP. A new marking scheme was introduced to detect the congestion. Instead of single threshold ‘K’, the DC-DCTCP uses two thresholds ‘K1’ and ‘K2’ as shown in Figure 3.2. When Queue length inside switch buffer increases and crosses threshold ‘K1’ the ECN marking starts ,and when Queue length inside switch buffer decreases and crosses threshold ‘K2’ the ECN marking stops. Although the authors of DT-DCTCP found out that by using two thresholds, it makes their proposed protocol more stable; still the problem of Incast with large number of senders was not resolved.

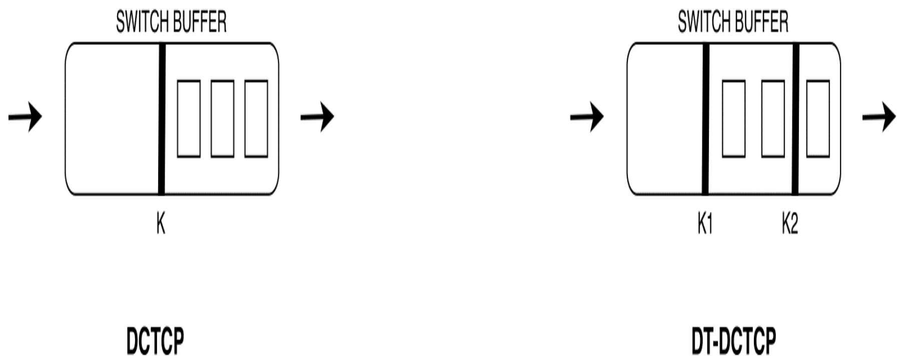


Figure 3.2: Comparison of threshold level used in DCTCP and DT-DCTCP.

3.1.4 ICTCP

Wu et al. [45] had proposed an algorithm named as Incast congestion control for TCP (ICTCP), which was applied on the receiver side. The receiver end does congestion control because it is able to know the available bandwidth and the throughput of all existing TCP connections. The receiver end is able to control the sender rate by changing the receiver window size. The main objective of this method is to decrease the packet loss instead of recovering after packet loss events. However, this technique has few weaknesses such as: it is not able to know the status of queue buildup inside switch buffer which is a very serious issue and also it achieves low throughput in case of less number of senders.

3.1.5 IDTCP

Wang et al. [46] proposed an algorithm to mitigate Incast is known as Incast Decrease TCP (IDTCP).This algorithm consist of three parts;

- It constantly monitors the congestion level of link by calculating minimum RTT and average RTT.
- It dynamically adjust the congestion window (cwnd) as per the congestion level of bottleneck link.
- It sets the cwnd = 1, if congestion link is fully congested.

However, continuously measuring the congestion level of the link is an exhaustive process. It achieves low throughput when number of senders are less.

3.1.6 PAC-TCP

Bai et al. [47] presented a protocol for DCNs known as Proactive Acknowledgement Control TCP (PAC-TCP) for managing the issue of TCP Incast at the receiver end. The ACK are not only used for received packets, but they are also used as a trigger for new sending packets. The algorithm proactively intercept and release the acknowledgement packets based on prior calculations. Its working is shown is Figure 3.3. However the success of algorithm is highly dependent on threshold value of switch; a small variation in switch threshold would lead to performance degradation.

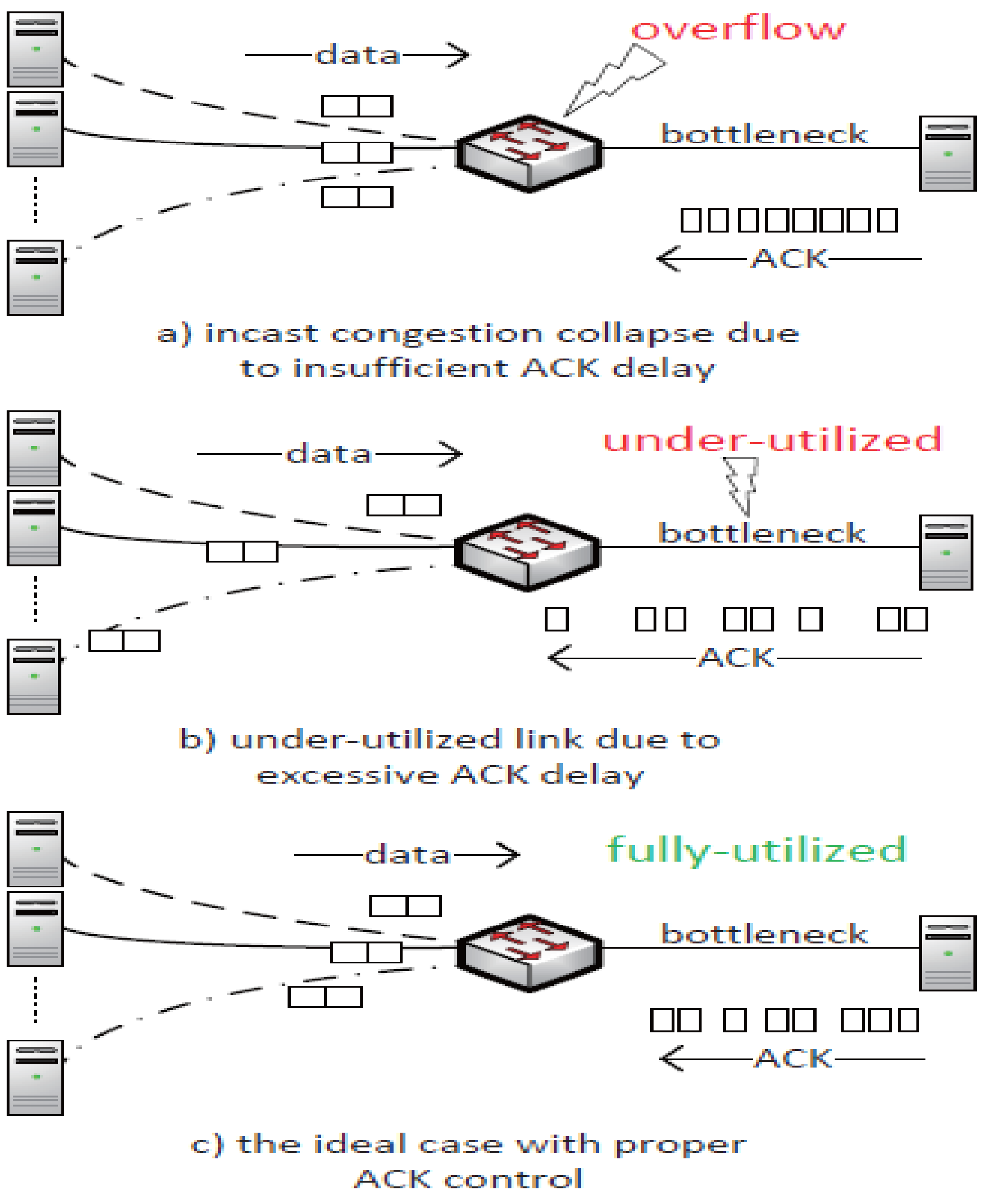

data
overflow
bottleneck
ACK
a) incast congestion collapse due to insufficient ACK delay
data
under-utilized
bottleneck
ACK
b) under-utilized link due to excessive ACK delay
data
fully-utilized
bottleneck
ACK
c) the ideal case with proper ACK control

Figure 3.3: The basic idea of working of PAC-TCP [47].

3.1.7 FATCP

A switch based method for improving throughput under TCP Incast scenario was proposed by Lee et al. [48]. A slight modification in explicit congestion notification (ECN) method was introduced. Proposed algorithm can be divided into two parts;

- Target flow is identified.
- Congestion control modification.

Its working is shown is Figure 3.4. The algorithm was limited to target small number of flows; also hardware modifications were required.

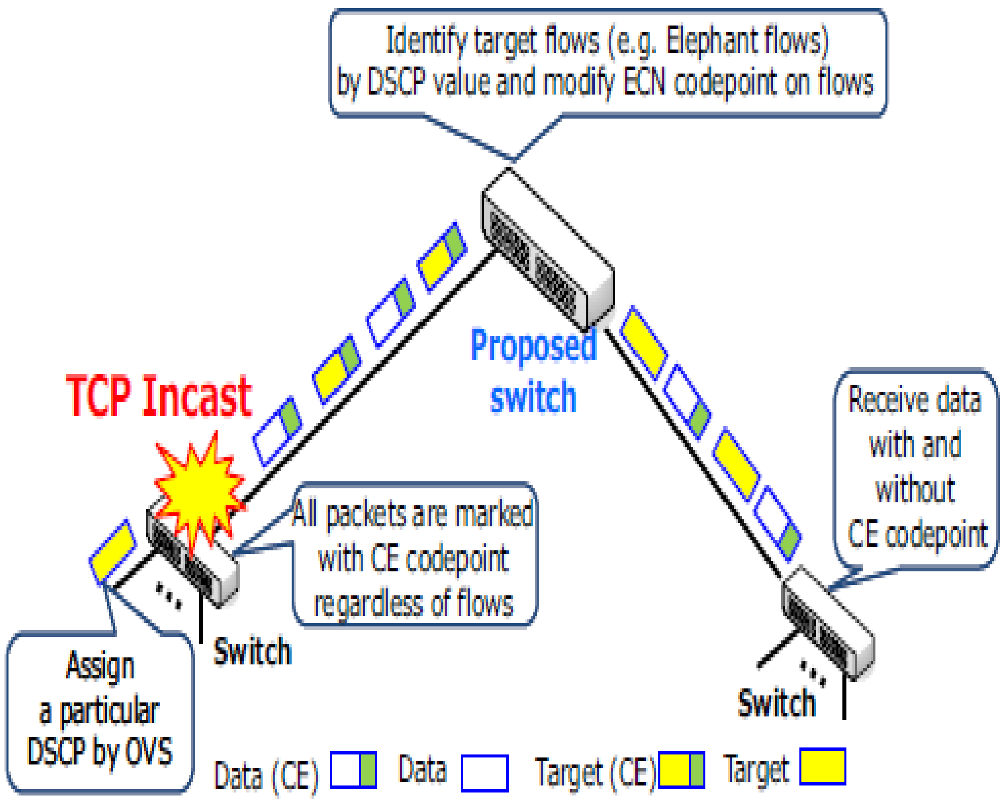

Identify target flows (e.g. Elephant flows) by DSCP value and modify ECN codepoint on flows
Proposed switch
TCP Incast
Receive data with and without CE codepoint
All packets are marked with CE codepoint regardless of flows
Switch
Switch
Assign a particular DSCP by OVS
Data (CE)
Data
Target (CE)
Target

Figure 3.4: The basic idea of working of FATCP [48].

3.2 Comparison Between Different Decentralized Incast Mitigation Techniques

Table 3.1 shows various decentralized Incast mitigation techniques along with their features and limitations.

Table 3.1: TCP Incast Handling Techniques for Decentralized Data Center Networks

Incast Solutions	Approach	Features	Limitations
Fined-grained RTO [21]	Distributed	Decreasing RTO to mitigate Incast	High precision timers are required for mitigating Incast problem in data centers, which is a very challenging task
DCTCP[31]	Distributed	Sender based congestion control scheme which uses ECN for congestion detection	Scalability issues
DT-DCTCP[44]	Distributed	Sender based Incast mitigation method which uses ECN with a different packet marking scheme to increase stability of DCTCP	Scalability issues
ICTCP[45]	Distributed	Receiver based Incast mitigation by updating ACK window	Tackling Incast scenario when the receiver and senders are not connected with the same switch is a major concern

IDTCP[46]	Distributed	Dynamically adjust the congestion window according to congestion level of bottleneck link	Bottleneck link is underutilized when concurrent senders are less and scalability is highly dependent on buffer size
PAC-TCP[47]	Distributed	Receiver based congestion control scheme which uses ECN for congestion detection	A slight variation in calculated parameters could lead to performance degradation
FAT-TCP[48]	Distributed	Switch based method for mitigating Incast	Identifying target flows when number of flows are large is a major challenge

3.3 Tackling Incast via Centralized Approach

In this section we will discuss the existing SDN based TCP Incast mitigation techniques for DCN.

3.3.1 OpenTCP

Ghobadi et al. [49] has proposed a framework named as OpenTCP. It uses an application Oracle which runs on the SDN controller to collect network statistics from the switch. The congestion control agents (CCA) are installed on the hosts. The Oracle distributes the policy to the CCA via switch. The policy is defined by the network administrator. CCA can choose between different TCP variants as per the defined policy. A schematic view of the OpenTCP working is shown in Figure 3.5. However, it is only a framework to tackle Incast, more work is required e.g. designing a proper congestion control policy. Also it needs extra care while gathering topology information and buffer statistics, otherwise it may increase the overhead.

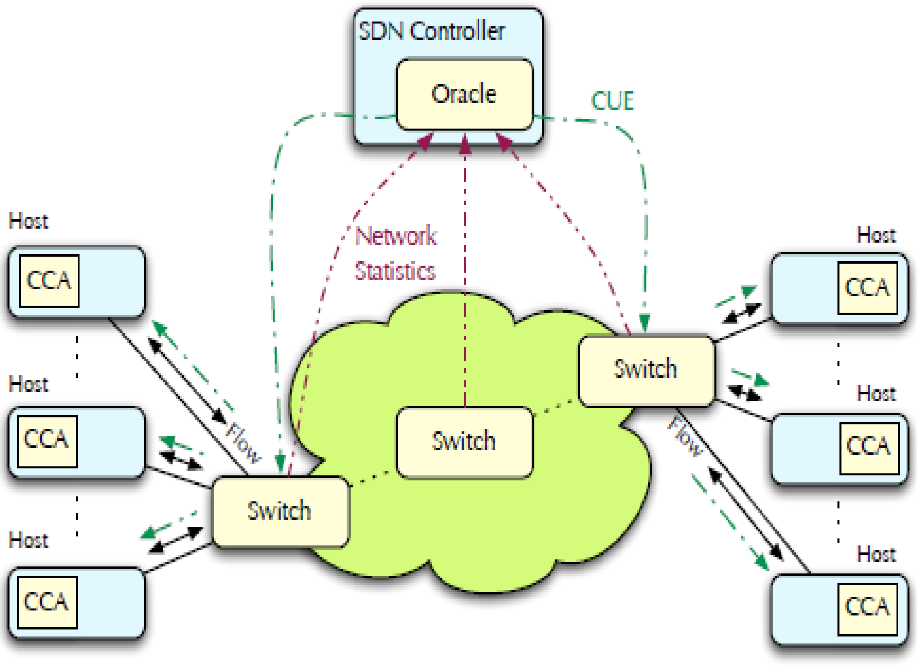


Figure 3.5: A schematic view of the OpenTCP working [49].

3.3.2 OTCP

An SDN based approach named as Omniscient TCP (OTCP) was proposed by Jouet et al. [50] to mitigate incast. OTCP collects various network parameters e.g. latency, buffer size and throughput to measure different TCP parameters such as maximum RTO, minimum RTO, maximum value of congestion window, and initial value of congestion window. These TCP parameters are tuned to control Incast. However, the calculation of these parameters causes enough overhead for the controller. Also elephant flows suffer from queue buildup.

3.3.3 SED

SED a software-defined approach was presented by Lu et al. [51] to solve the Incast problem in DCN. It divides the flows into two types: deadline and non-deadline flows. The non-deadline flows are throughput sensitive while the deadline flows are delay sensitive. Congestion process is detected by the switch as it signals the congestion to the controller if switch buffer overflows. The non-deadline flows are always given a sending rate of 1 MSS. Global information flow table (GIF) is maintained by the controller which is used to keep the record of flow size, flow identification number, deadline etc. The table is sorted according to the deadline. The basic idea of working of SED is shown in Figure 3.6. However the goodput of SED decreases sharply as the number of senders are more than 40. Here we argue that SED also require to store and process such a comprehensive table (GIF) which causes significant overhead for the controller, the table maintained by the controller should be precise and less complex in calculations.

3.3.4 SDTCP

Lu and Zhu [52] proposed SDTCP which aims to achieve high throughput, low latency, and better burst tolerance. The congestion is detected when the queue built up inside SDN switch crosses the threshold. Upon detection of the congestion, SDN switch notifies the controller. The controller instructs the switch to modify the ACK window

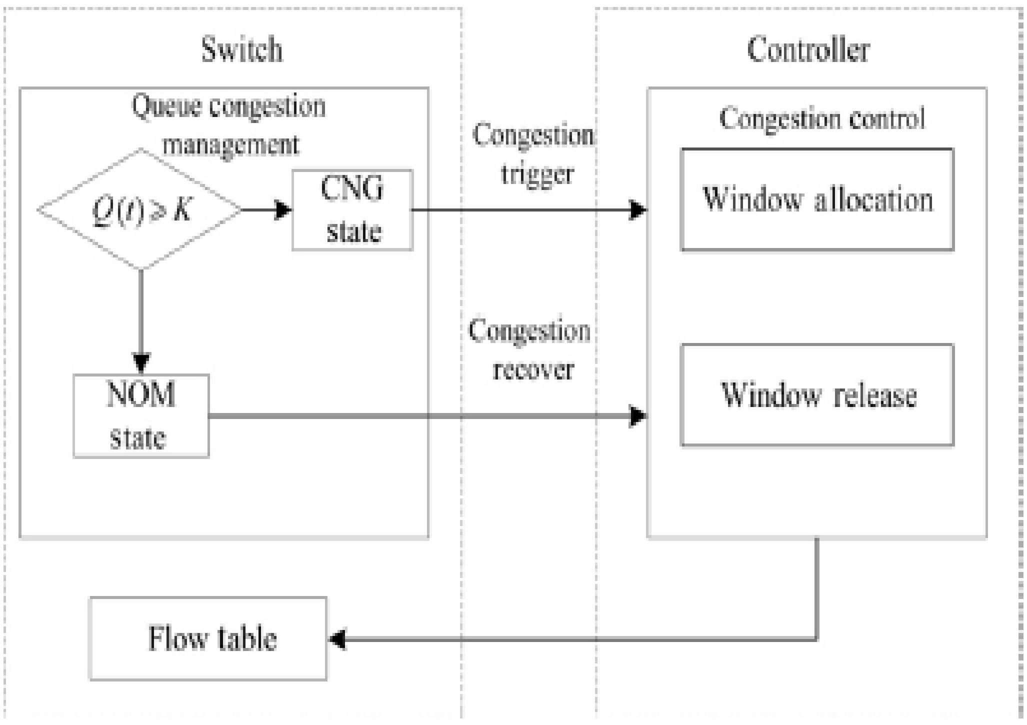


Figure 3.6: The basic idea of working of SED [51].

(awnd) in order to curb the congestion. The proposed mechanism in aimed to target the elephant flows and create room for mice flows. A modified version of SDTCP was also presented by Lu et al. [53] in which there are three different threshold levels upon which three types of congestion notifications are generated such as low level congestion notification, medium level congestion notification and high level congestion notification. SDN controller give instructions to the SDN switch to change the awnd as per received congestion notification. Experimental results show SDTCP performs better than DCTCP but still there is no mechanism to deal with the starvation of elephant flows.

3.3.5 SIG

Xu et al. [54] presented a scheme, SIG which splits the senders into smaller groups and then allow each group to transmit data at a time. SIG mechanism is call on by the SDN controller when many senders want to send requests to a single receiver. However, it has some scalability issues because it only allow 30 senders to transmit data at a time.

3.3.6 SICCQ

A centralized approach for mitigating Incast using queue based monitoring (SICCQ) was presented by Abdelmoniem et al. [55, 56]. The controller keeps track of TCP SYN/FIN flags and calculate a weighted moving average of the buffer occupancy upon which it predicts that the congestion will occur. If the queue length exceeds the certain threshold the controller sends an INCAST ON message to the hypervisor. The hypervisor starts rewriting the ACK window to 1 MSS/RTT. When buffer queue occupancy decreases below 20 percent threshold, an INCAST OFF message is sent by the controller which is intercepted by the hypervisor to stop modifying the ACK window. Thus senders start regaining there original sending rates. However, continuously measuring the buffer occupancy and predicting Incast is a cumbersome process for the SDN controller. The INCAST ON and INCAST OFF messages are increasing overhead because the controller is continuously communicating with the hypervisor, also the hypervisor based solutions are better in virtualized environment.

3.3.7 DyFaShBuP

Bangash et al. [32] presented a centralized approach named as DyFaShBuP to mitigate Incast in storage clusters. The SDN controller dynamically calculates the total number of attached storage servers based on IP or MAC addresses. According to the network conditions and storage servers the SDN controller makes a policy, distributes it among all storage nodes via OpenFlow switches. The sender thus adjust their sending rate according to the received policy. Our proposed method is also based on DyFaShBuP with a slight modification in the process for calculating storage nodes.

3.4 Comparison Between Different Centralized Incast Mitigation Techniques

Table 3.2 shows various centralized Incast mitigation techniques along with their features and limitations.

Table 3.2: TCP Incast Handling Techniques for Centralized Data Center Networks

Incast Solutions	Approach	Features	Limitations
Open TCP[49]	Centralized	An automated framework for tuning TCP which can choose between different TCP variants as per the defined policy	Computation Overhead
OTCP[50]	Centralized	Mitigate Incast by tuning various TCP parameters	Unable to solve Incast problem with large flows
SED[51]	Centralized	Mitigating Incast by sorting a GIF table on basis of flow deadline	Computation Overhead
SDTCP [52], [53]	Centralized	Controller uses ECN enabled switch for detecting congestion and updating ACK window	Scalability Issues
SIG[54]	Centralized	Controller limits the number of senders by dividing all senders into smaller group and allowing one group at a time	Scalability Issues

SICCQ [55], [56]	Centralized	Controller predicts congestion and signals the hypervisor for updating the ACK window	Computational Complexity
DyFaShBuP [32]	Centralized	SDN Controller formulates and distributes the policy to the sending servers for adjusting their sending rate	No details about, how IP or MAC address is used for calculating number of storage nodes

CHAPTER 4

Architecture and Proposed Method

We proposed a technique, modified dynamic fair-share buffer policy (M-DyFaShBuP) for mitigating Incast. It not only resolves the TCP throughput collapse problem, but also it completely utilizes the bottleneck link capacity. Our aim is to design an algorithm for the SDN controller, which simplify the calculation on controller. The SDN controller is the main entity in our proposed method, which makes and distributes the policy to the servers.

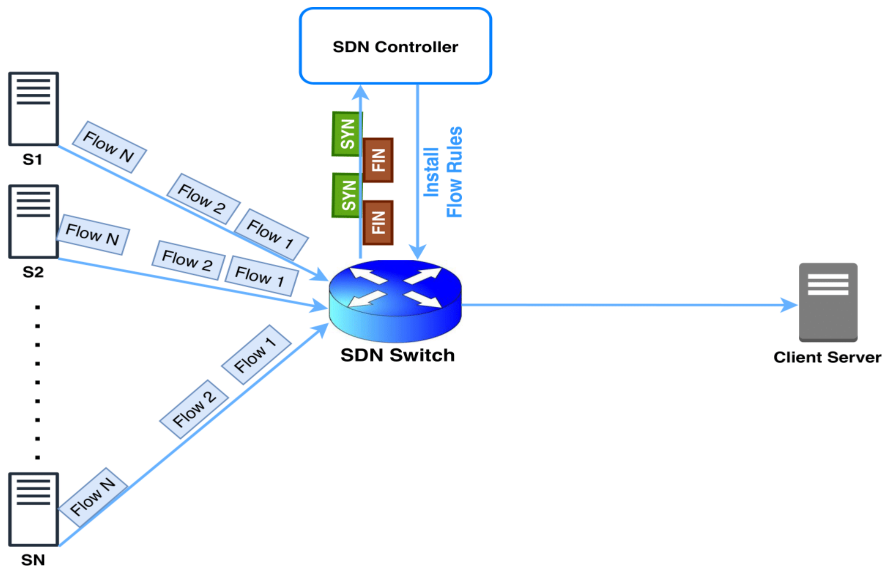


Figure 4.1: A normal process of communication in data centers: Storage servers send data, in the form of flows, simultaneously to the client server; SDN controller install flow rules in switch.

A normal method of communication inside a cluster based storage system is shown in

Figure 4.1: The storage servers transmit their portion of data to the client server via flows and the controller installs flow rules in the switch. These flows combine to produce the application data [57]. We implemented our proposed technique by utilizing the programmable nature of SDN controller, which can operate in many-to-one communication scenario.

4.1 Design Rationale

In contemporary data centers, TOR design and EOR design are mostly used. In TOR design, all the servers in a single rack are connected with a switch that is mounted on top of rack. In EOR design the servers in an entire row, which include all racks are directly connected to a switch placed at the corner of each row [58]. Many of the methods used earlier for controlling Incast have only considered TOR design. In our approach, we considered both TOR and EOR designs.

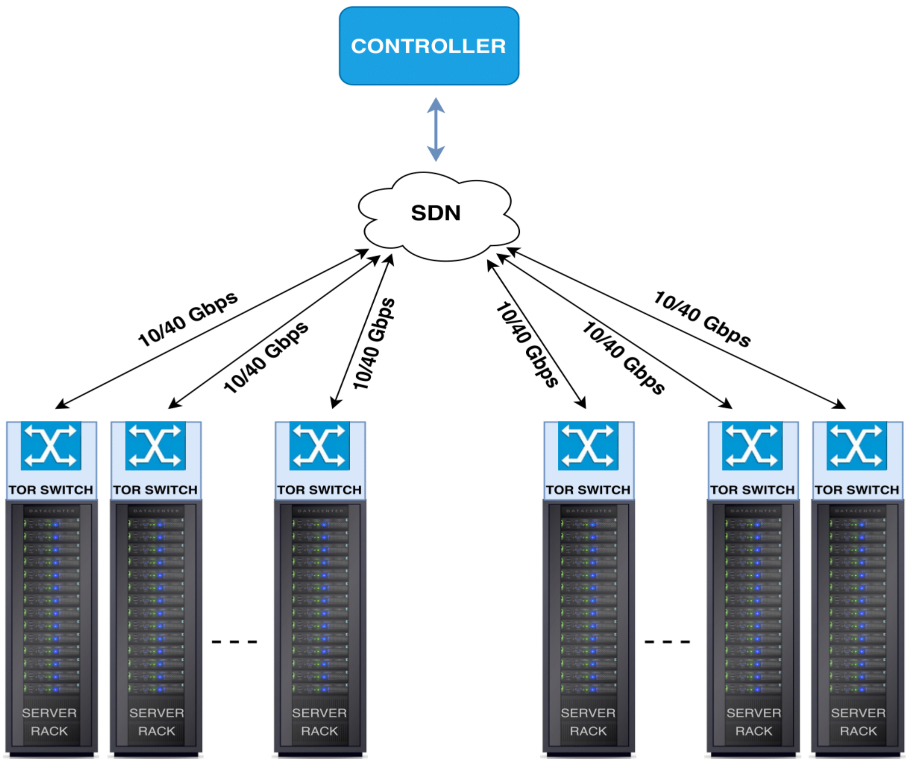


Figure 4.2: TOR approach for data centers: Server to switch link is 1 Gbps and switch to switch link is 10/40 Gbps.

Figure 4.2 depicts the TOR design, where all servers in a rack are connected to a switch with 1/10 Gbps link and all switches are interconnected with a 10/40 Gbps link. While Figure 4.3 depicts the EOR design, where servers in an entire row are connected to single switch with 10 Gbps link and all switches are interconnected with a 10/40 Gbps.

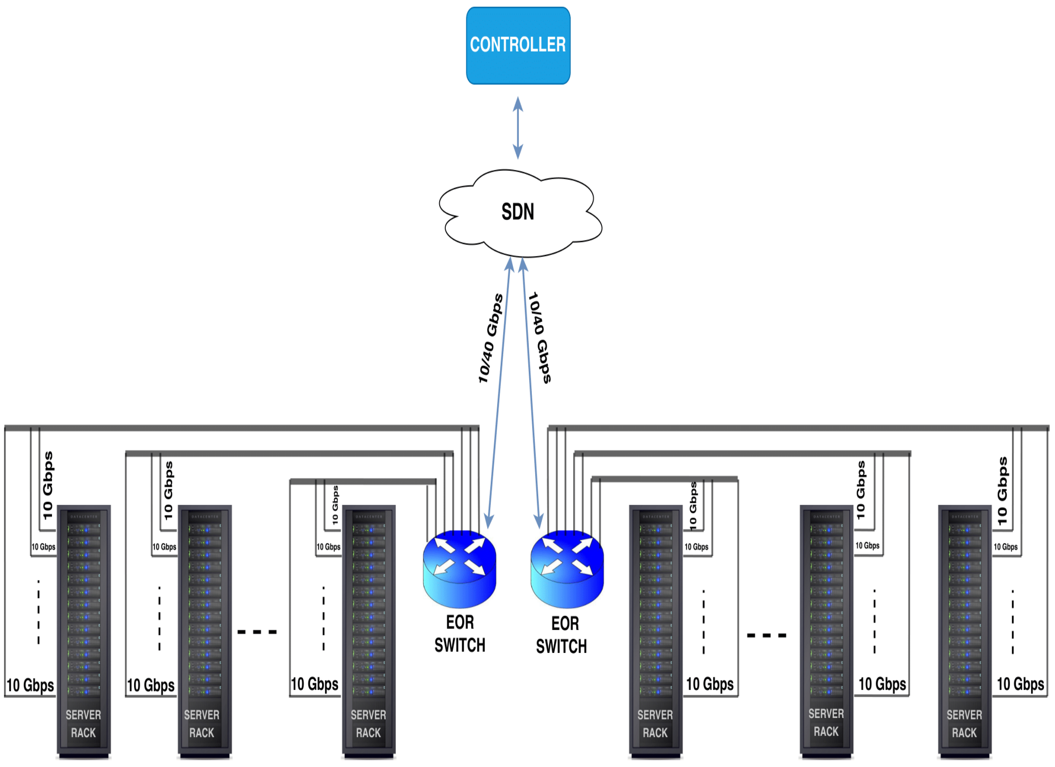


Figure 4.3: EOR approach for data centers: Server to switch link is 10 Gbps and switch to switch link is 10/40 Gbps.

4.1.1 Role of SDN Controller

The controller plays an important role in our design while mitigating Incast. It dynamically counts the total servers involved with the help of TCP SYN/FIN counters. The separate flow counters are maintained by the SDN controller for each server. As per the total number of servers in active state, the SDN controller calculates a fair-share for each server. Observing the network statistics and the number of servers in active state, the controller devise a policy and distribute it among all servers as shown in Figure 4.4.

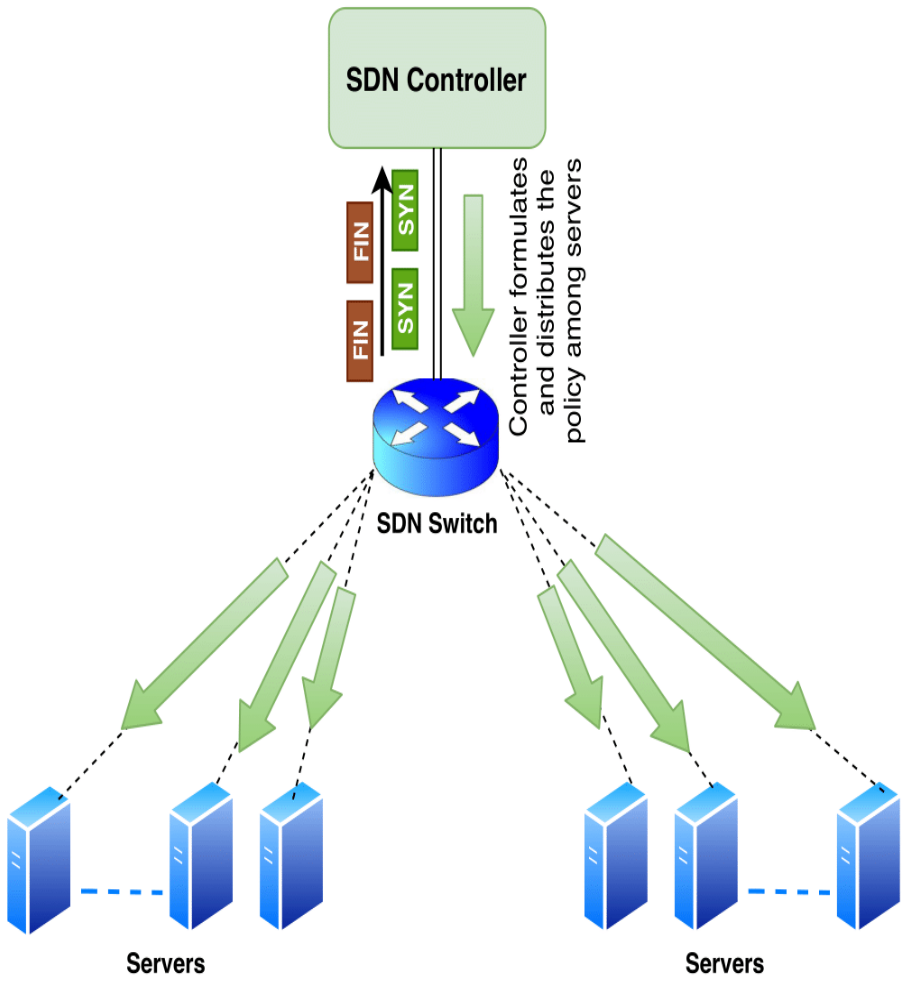

SDN Controller
FIN
SYN
FIN
SYN
Controller formulates and distributes the policy among servers
SDN Switch
Servers
Servers

Figure 4.4: SDN Controller is responsible for implementing and distributing the policy among all storage servers.

4.1.2 SDN Switch

In our proposed method, the switch has the minimum role to play while tackling Incast. We eliminated the conventional ECN mechanism, which was used earlier for congestion detection. It not only decreases the cost, but it also reduces the burden on SDN switch. We programmed the switch with a simple Copy-to-Controller rule for SYN/FIN packets [59]. The switches will send a copy of SYN/FIN packet towards controller. We considered a TOR switch having 48 ports and 1 GB Ethernet interface for each port and an EOR switch with 128 ports and 10 GB Ethernet interface for each port. Our design enable expansion; hence, in future the network can be upgraded from 1/10 GE to 10/40 GE.

4.1.3 Storage Servers

We have incorporated most commonly used Rack servers in our design. 48 servers are placed in a single rack. We have established a 1 GB Ethernet link for each server when they are connected with a TOR switch and a 10 GB Ethernet link for each server when they are connected with an EOR switch. These servers are installed with a software agent. The agent updates the window size of servers according to the policy it receives from the controller.

4.1.4 Bandwidth Delay Product

The bandwidth delay product (BDP) decides the total amount of data that can be sent on a link in a given time period. In data centers the value of RTT is 1 ms. This is the main reason due to which, links are not able to support large amount of data when multiple servers are sending data to a single receiver. During Incast scenario, when

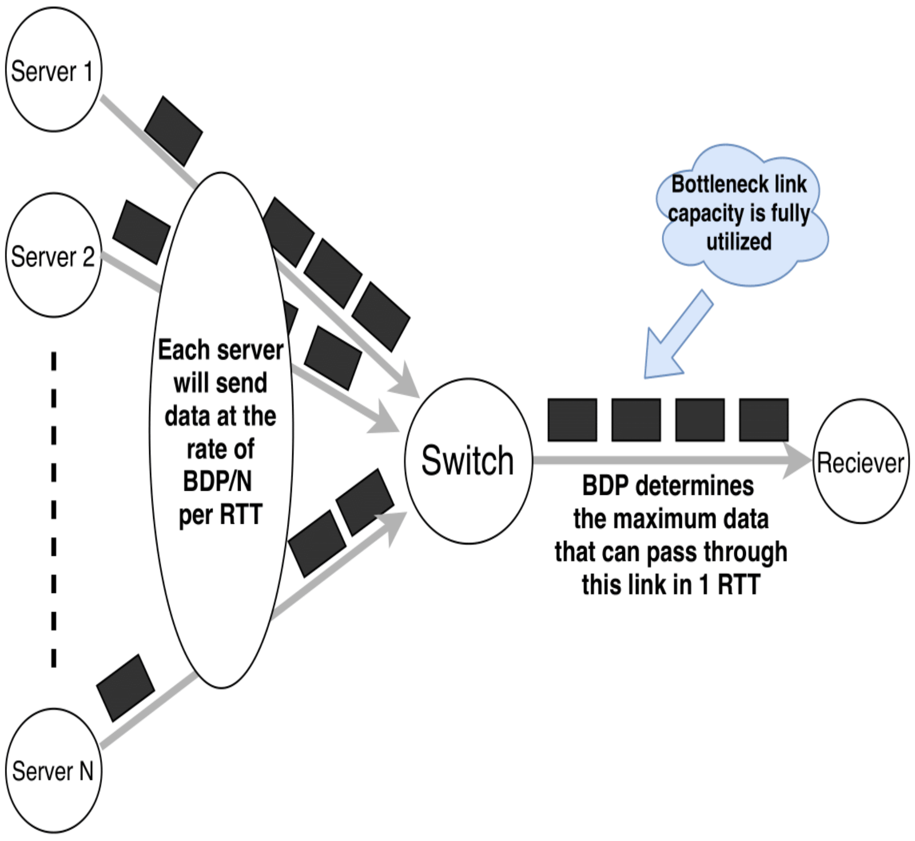

Server 1
Server 2
Server N
Each server will send data at the rate of BDP/N per RTT
Bottleneck link capacity is fully utilized
Switch
Reciever
BDP determines the maximum data that can pass through this link in 1 RTT

Figure 4.5: Receiver BDP limits the rate of sending servers to avoid Incast.

the aggregate BDP of multiple sending servers' link becomes more than the BDP of the receiver's links; the receiver's link becomes a bottleneck. Figure 4.5 shows that for avoiding Incast, we have to control the sending rate of servers to a limit, where they never exceeds the BDP of a single receiver's link. This is the main reason, we have utilized BDP in our method for calculating the equitable share for each server.

4.2 M-DyFaShBuP

According to the design rationale discussed above we formulated M-DyFaShBuP. We have revised the process by which the number of servers were calculated in [32]. We presented an algorithm that can dynamically calculate the number of servers using a separate SYN/FIN counter for each server. The controller maintain these counters and it only requires to store an integer which is used to represent the total number of flows generated per server. Earlier in [51], per flow statistics such as min RTT, max RTT etc. were required by controller for handling congestion. This not only increased the overhead but it also increased the computational complexity for the controller. In our proposed algorithm we only require TCP SYN/FIN packets which are flow initiation and flow termination packets. These packets are used by the controller for implementation of policy. The proposed algorithm is shown below and the parameters used in this algorithm are described in the Table 4.1.

The algorithm starts as the client server requests a file from multiple servers as shown in Figure 4.6 and the SDN controller installs the Copy SYN/FIN rule in the switch as shown in Figure 4.7. The file request is sent to all the servers on which the file is distributed.

The mechanism of M-DyFaShBuP algorithm is divided into following steps;

- Switch will forward all TCP SYN/FIN packets to the SDN controller.
- If the received packet has a SYN bit flag ON, then the controller will increment the flow counter by 1.
- If the received packet has a FIN bit flag ON, then it will decrement the flow

Algorithm 1: Algorithm for SDN Controller

```
Input: $P$
Output: $Fs$
1  Install Copy SYN/FIN Rule in SDN Switch  Function ($P$):
2      if $SYN - flag(P)$ then
3          Cs = Cs + 1 ;
4      end
5      if $FIN - flag(P)$ then
6          Cs = Cs - 1 ;
7      end
8      return $Cs$
9  End Function
10 Function ($Cs$):
11     NA = Number of Active Cs Counters ;
12     return $NA$
13 End Function
14 Function ($NA$):
15     BDP = Cp * RTT;
16     Fs = ( BDP)/ NA ;
17     if $NA - changes;$ then
18         Update Fs ;
19         Distribute Fs to Attached Servers ;
20     end
21     return $Fs$
22 End Function
```

Table 4.1: Table of parameters for M-DyFaShBuP Algorithm

Parameters	Description
N	Total number of Source Servers
Cs	Set of Storage Servers Counter = [CS1, CS2,..,CSN]
P	Packet (SYN or FIN)
NA	Total number of Active Counters
Cp	Bottleneck Link Capacity
RTT	Common Round Trip Time
Fs	Fair-share for each Server
BDP	Bandwidth Delay Product

counter by 1.

- All servers will have a separate TCP SYN/FIN counter, that is maintained by the SDN controller.
- The controller will count the total number of counters which are in active state (NA).
- Now, the controller will calculate fair-share by using the following formula;
  BDP = Cp * RTT;
  Fs = BDP/ NA;
- It will distribute the fair-share to all servers.
- The controller will update the fair-share, whenever the number of active counter changes.

M-DyFaShBup is a controller based technique for tackling Incast. The controller plays the principal role from policy making till policy implementation. Figure 4.8 shows a schematic view of M-DyFaShBuP architecture. The figure explains the entire process of M-DyFaShBuP. It shows that the multiple storage servers send their portion of data in the form of flows, while responding to a client's request. The TCP SYN/FIN packets

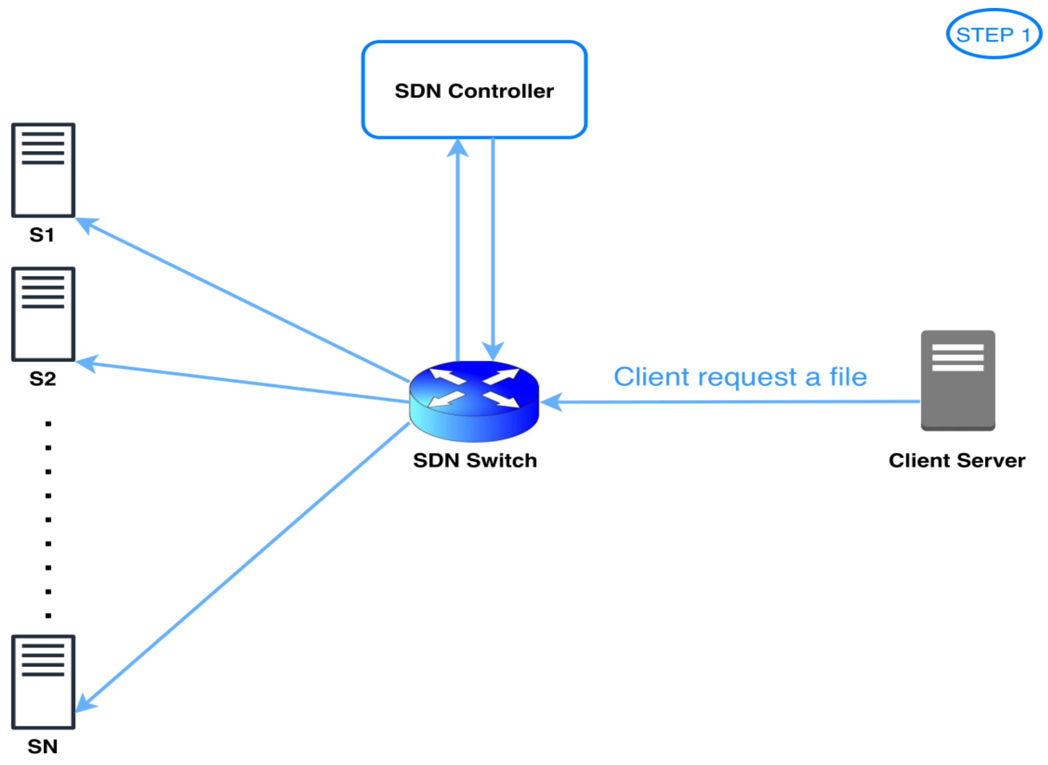


Figure 4.6: Client server requests a file from multiple servers.

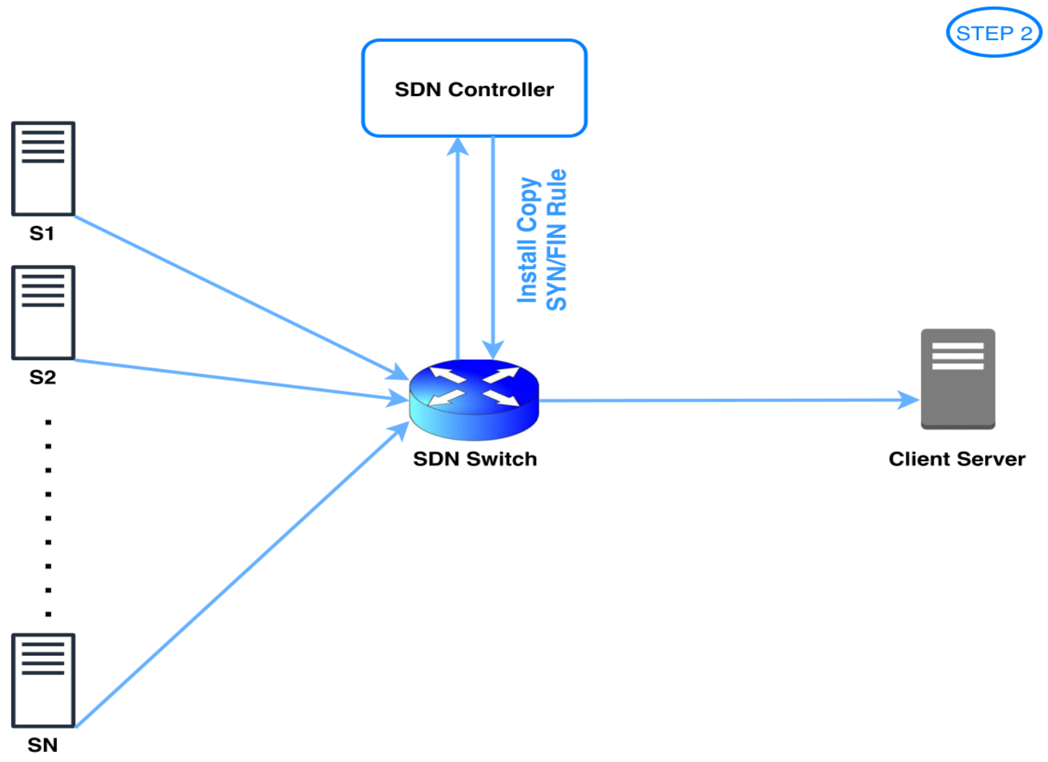
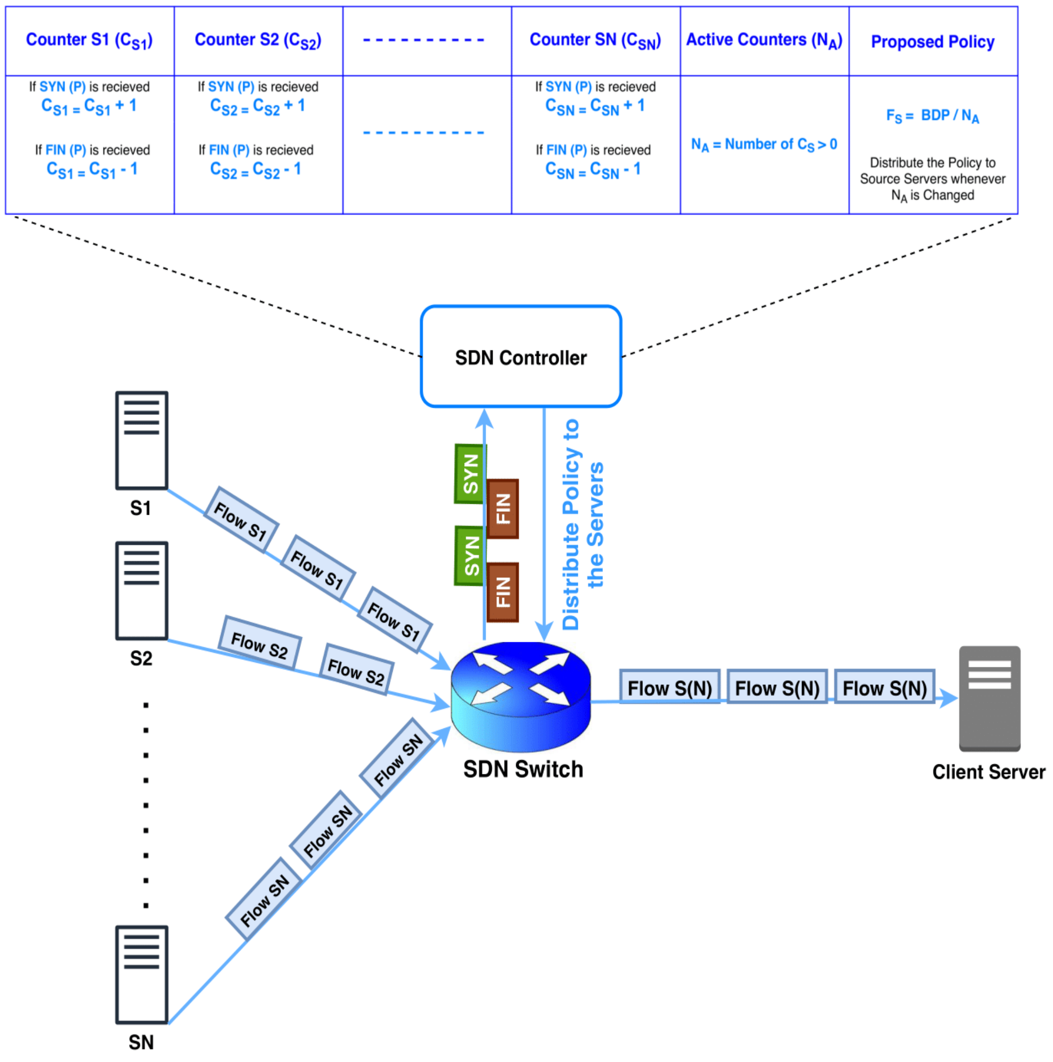


Figure 4.7: SDN controller installs the Copy SYN/FIN rule in the switch.

are forwarded to the controller. The SDN controller has maintained a separate flow counter for each server. An equitable share for each server is calculated by counting the number of counters which are active. SDN controller makes and distributes the policy to all storage servers. The servers modify their sending rates as per the policy they received by the controller.



Counter S1 (C_{S1})	Counter S2 (C_{S2})	----------	Counter SN (C_{SN})	Active Counters (N_A)	Proposed Policy
If SYN (P) is recieved $C_{S1} = C_{S1} + 1$ If FIN (P) is recieved $C_{S1} = C_{S1} - 1$	If SYN (P) is recieved $C_{S2} = C_{S2} + 1$ If FIN (P) is recieved $C_{S2} = C_{S2} - 1$	----------	If SYN (P) is recieved $C_{SN} = C_{SN} + 1$ If FIN (P) is recieved $C_{SN} = C_{SN} - 1$	N_A = Number of $C_S > 0$	$F_S = BDP / N_A$ Distribute the Policy to Source Servers whenever N_A is Changed

Figure 4.8: Schematic diagram of M-DyFaShBup architecture: The controller formulates and distributes the policy to all storage servers.

4.3 Avoiding Incast for IoT Applications via M-DyFaShBuP

The instances are hosted on the data center infrastructure [60], that can be a same physical machine, or on different machines in same rack, or on different machines in different racks. The tenants buy VMs on rent for deploying their IoT Applications. Because of large amount of data influx, efficient data communication is required among the machines in a data center. Hence, it is important that the protocols used for data communication are chosen carefully, otherwise it could result in increased latency or reduced throughput.

The process for managing the traffic in a data center have a significant impact on the performance of IoT services. The TCP Incast results in increase latency, that can affect many business activities. Moreover, under Incast scenario the data centers are not able to handle large number of IoT applications. Hence such a situation can impact the revenue of business organizations. Therefore, we proposed M-DyFaShBuP for avoiding Incast in cloud data centers and for supporting large number of IoT applications as depicted in Figure 4.9. An efficient cloud solution can give rise to the revenue of business organizations. M-DyFaShBuP provides high throughput for supporting multiple IoT services.

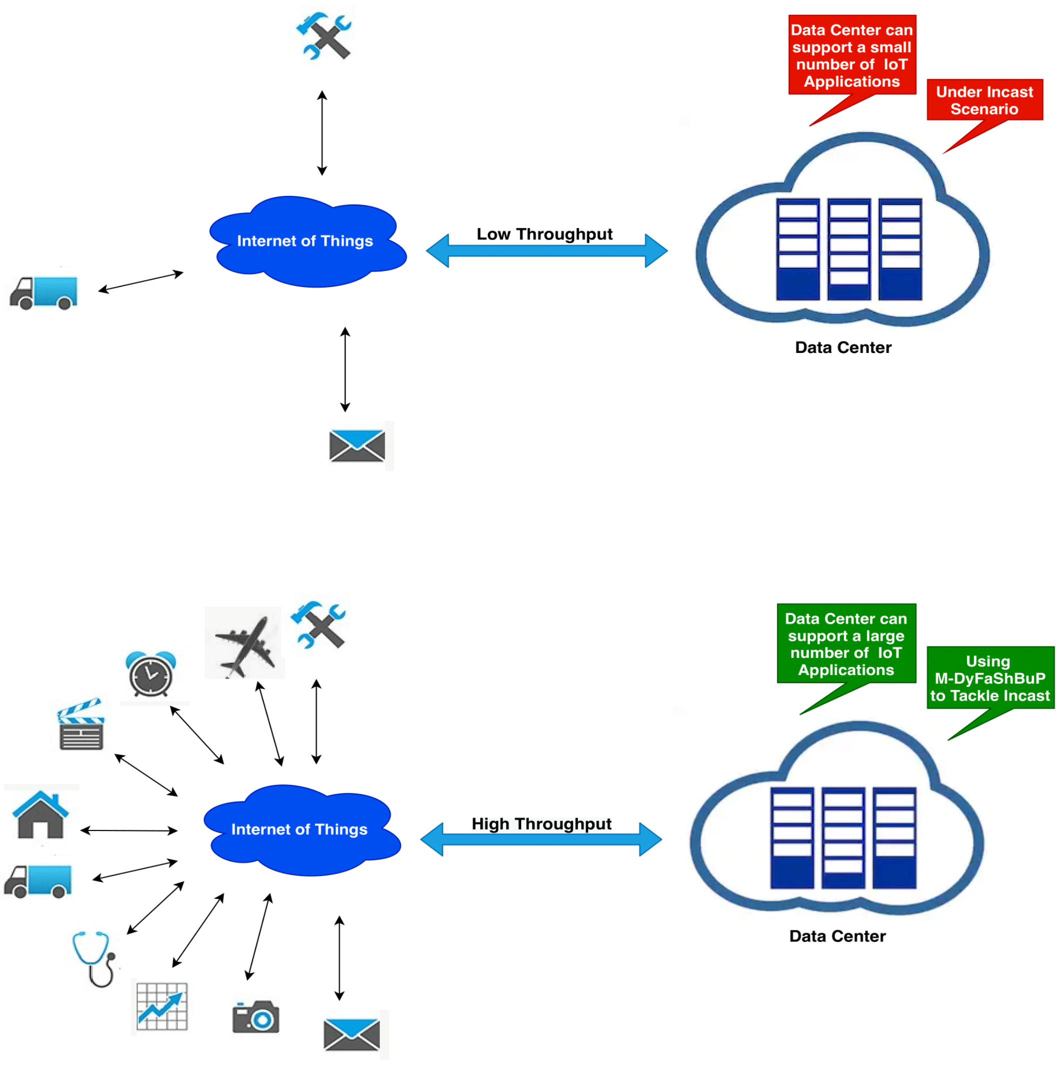

Data Center can support a small number of IoT Applications
Under Incast Scenario
Internet of Things
Low Throughput
Data Center
Data Center can support a large number of IoT Applications
Using M-DyFaShBuP to Tackle Incast
Internet of Things
High Throughput
Data Center

Figure 4.9: IoT and cloud data centers: During Incast, less number of IoT applications are supported while avoiding Incast via M-DyFaShBuP increases the number of IoT applications.

CHAPTER 5

Experimental Results and Analysis

We used MATLAB simulation environment for calculating our results. We have included **TOR** and **EOR** designs in our experimental setups. The scenarios which we have created in our experimental setups, considered a client server requesting a file distributed across multiple servers. In our results, we have shown a comparison between the numbers of servers and bottleneck link throughput, as the increase in number of servers plays an important role in causing Incast. .

5.1 Experimental Setup for TOR Design

Figure 5.1 shows the experimental setup for TOR design. All the servers in a rack are connected to a single TOR switch, placed on top of rack with a 1 Gbps link and all switches are networked via a 10 Gbps link. Table 5.1 shows various parameters that are used in experimental setup for TOR design. We have developed two types of scenarios while calculating the results;

- An **intra-rack scenario**: Client server requests an **ebook**.
- An **inter-rack scenario**: Client server requests a **documentary**.

Most of the earlier work on Incast have only considered an intra-rack scenario. We have also considered an inter-rack scenario which is more realistic and it has increased the scope on this work.

Table 5.1: Table of parameters used in TOR Simulation

Parameters	Value
Number of Servers (intra-rack)	30 (can increase)
Number of Servers (inter-rack)	60 (can increase)
Number of Controllers	1
Number of Switches	Multiple
Bottleneck link Capacity	1 Gbps
Switch to Switch Bandwidth	10 Gbps
Buffer Capacity	150 KB
RTT (intra-rack)	0.6 ms
RTT (inter-rack)	0.8 ms

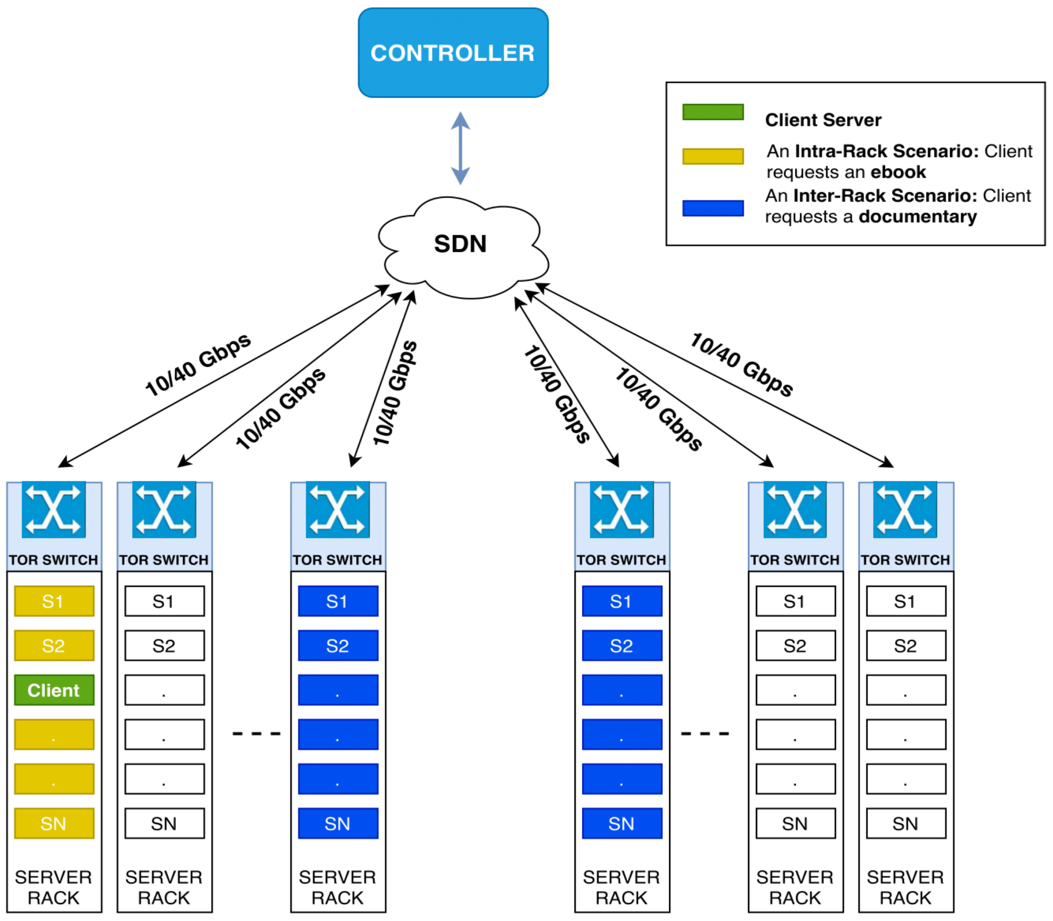

CONTROLLER
SDN
Client Server
An Intra-Rack Scenario: Client requests an ebook
An Inter-Rack Scenario: Client requests a documentary
10/40 Gbps
TOR SWITCH
S1
S2
Client
SN
SERVER RACK

Figure 5.1: Experimental Setup for TOR design incorporating different scenarios.

5.2 Experimental Results for TOR Design

5.2.1 An Intra-Rack Scenario: Client Server Requests an Ebook

Suppose a client requests an ebook that is stored across 30 server. In this scenario the client and the storage servers are present in a same rack. let the RTT for this scenario be 0.6 ms. All servers including client are connected to same TOR switch with a 1 Gbps link. The client's link BDP = 0.6ms * 1 Gbps, which is equal to 75 KB. It is equally divided among all 30 servers and a single server gets a 2.5 KB fair-share, if all 30 servers are sending data. Figure 5.2 shows that the bottleneck link capacity is fully utilized even if the number of sending servers are changing. The client server successfully receives the requested file from multiple storage servers without having any link degradation.

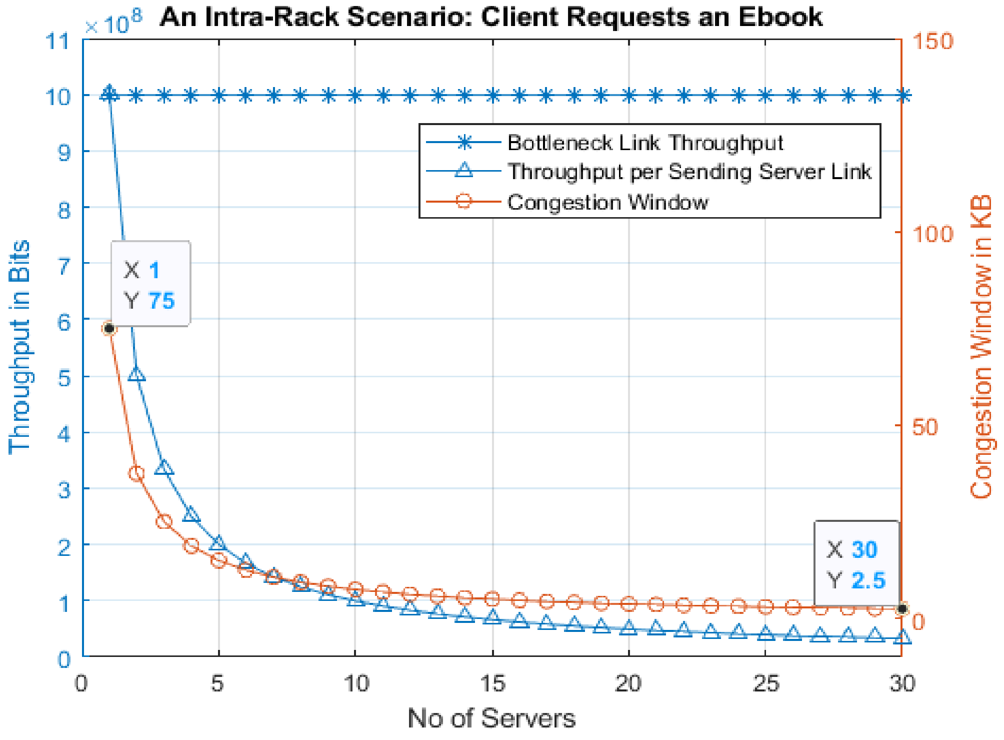


Figure 5.2: Throughput versus numbers of servers when a client requests an ebook in an intra-rack arrangement.

5.2.2 An Inter-Rack Scenario: Client Server Requests a Documentary

In this scenario we supposed a client requesting a documentary that is stored across 60 server. The client server and the storage servers are present in different racks. let the RTT for this scenario be 0.8 ms. All servers including client are connected to a TOR switch placed on different racks with a 1 Gbps link. The client's link BDP = 0.8ms * 1 Gbps, which is equal to 100 KB. It is equally divided among all 60 servers and each server gets a 1.66 KB fair-share, if all 60 servers are transmitting data. Again Figure 5.3 shows a similar behaviour such as the bottleneck link capacity is fully utilized even if the number of sending servers are increasing. Again the client server successfully receives the requested file from multiple storage servers without having any link degradation.

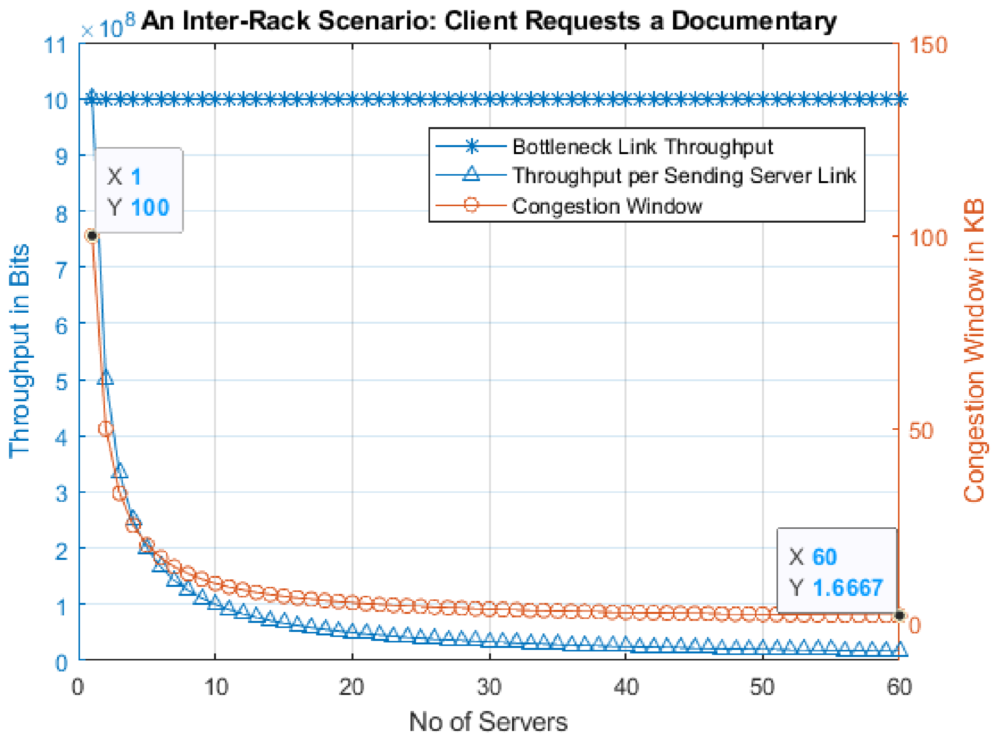


Figure 5.3: Throughput versus numbers of servers when a client requests a documentary in an inter-rack arrangement.

5.2.3 Analysis

In TOR design, we considered two scenarios in which a client is requesting an ebook and a documentary that is stored across multiple servers in an intra-rack and an inter-rack arrangements respectively. It can be observed, when a client requests an ebook, the minimum value of fair-share for a single server is 2.5 KB, which means enough window size is available that can be divided further to increase the number of servers. The maximum value of fair-share is equal to 75 KB which is actually same as the client's link BDP. It is still less than the switch buffer capacity. It means that the client's link is the bottleneck. Likewise, in other scenario, when a client requests a documentary that is distributed on 60 servers, the minimum value of fair-share is 1.66 KB, again it shows that number of servers can increase. The maximum fair-share is 100 KB that is again less than switch buffer capacity; hence ensuring that switch buffer is not the bottleneck. Figure 5.4 shows the comparison between both scenarios. In this figure, the distribution of BDP is shown via a 3D diagram.

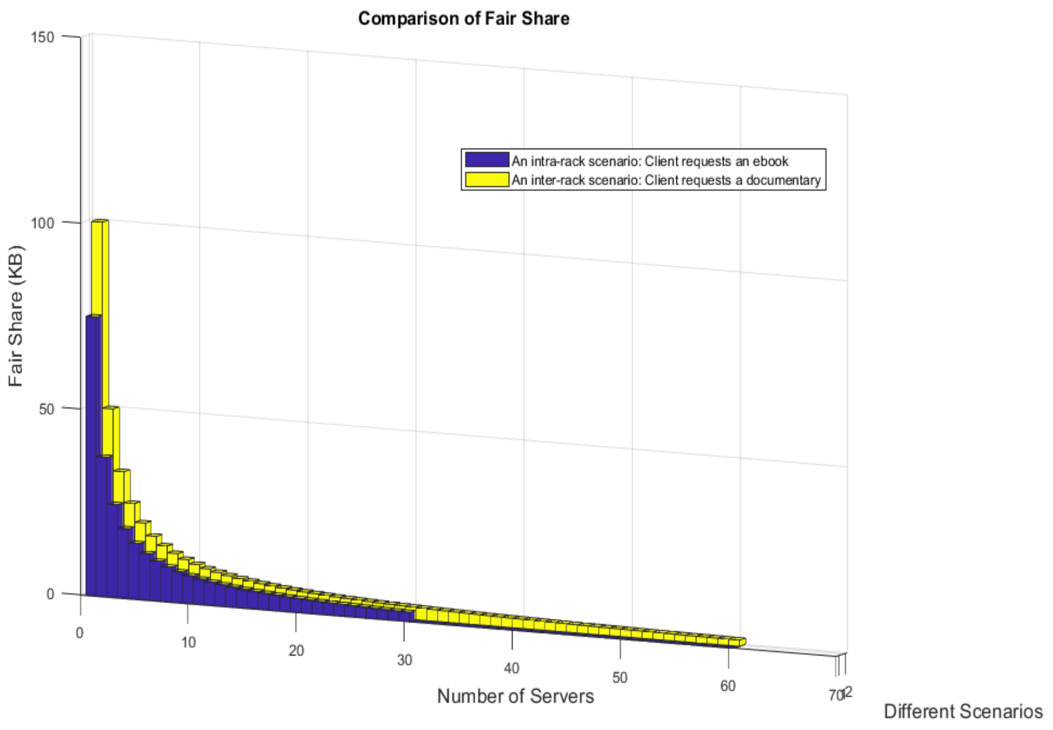


Figure 5.4: Fair-share vs number of servers: Comparison of BDP between different scenarios used in TOR setup.

5.3 Experimental Setup for EOR Design

Figure 5.5 shows the experimental setup for EOR design. All the servers in an entire row including all racks are connected to a single EOR switch, placed at the end of each row with a 10 Gbps link and all switches are networked together via a 10/40 Gbps link. Table 5.2 shows various parameters that are used in experimental setup for EOR design. Again we have developed two types of scenarios while calculating the results;

- An **intra-row scenario**: Client server requests an **annual report**.
- An **inter-row scenario**: Client server requests a **movie**.

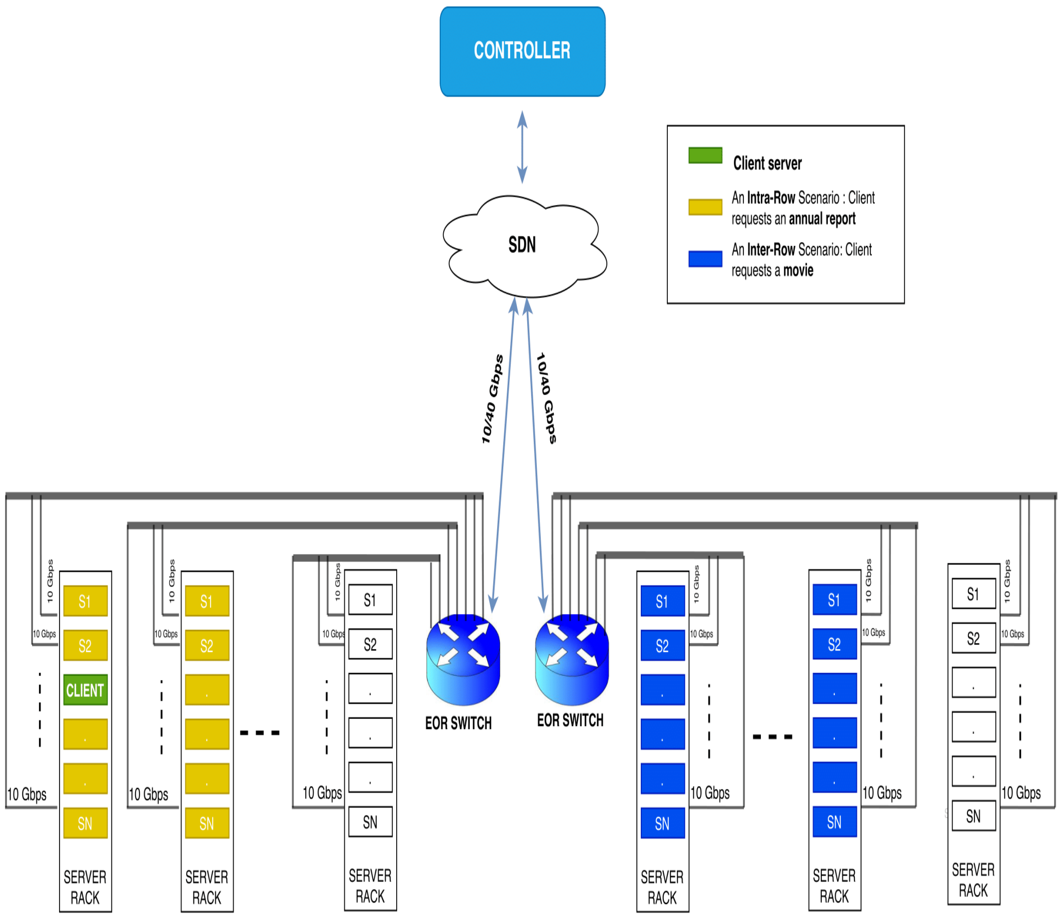

CONTROLLER
SDN
Client server
An Intra-Row Scenario : Client requests an annual report
An Inter-Row Scenario: Client requests a movie
10/40 Gbps
10/40 Gbps
EOR SWITCH
EOR SWITCH
S1
S2
CLIENT
SN
SERVER RACK
10 Gbps

Figure 5.5: Experimental Setup for EOR design incorporating different scenarios.

Table 5.2: Table of parameters used in EOR Simulation

Parameters	Value
Number of Servers (intra-row)	100 (can increase)
Number of Servers (inter-row)	150 (can increase)
Number of Controllers	1
Number of Switches	Multiple
Bottleneck link Capacity	10 Gbps
Switch to Switch Bandwidth	10/40 Gbps
Buffer Capacity	2 MB
RTT (intra-row)	1 ms
RTT (inter-row)	1.2 ms

5.4 Experimental Results for EOR Design

5.4.1 An Intra-Row Scenario: Client Server Requests an Annual Report.

Similarly, we supposed a client requesting an annual report that is stored across 100 server. In this scenario the client and the storage servers are present in the same row but might be in different racks. let the RTT for this scenario be 1 ms. All servers including client are connected to same EOR switch with a 10 Gbps link. The client's link BDP = 1 ms * 10 Gbps, which is equal to 1.25 MB. It is equally divided among all 100 servers and a single server gets a 12.5 KB fair-share, if all 100 servers are sending data. Again it can be observed from Figure 5.6 that the bottleneck link capacity is fully utilized even if the number of sending servers are changing. The client server successfully receives the requested file from multiple storage servers without causing Incast.

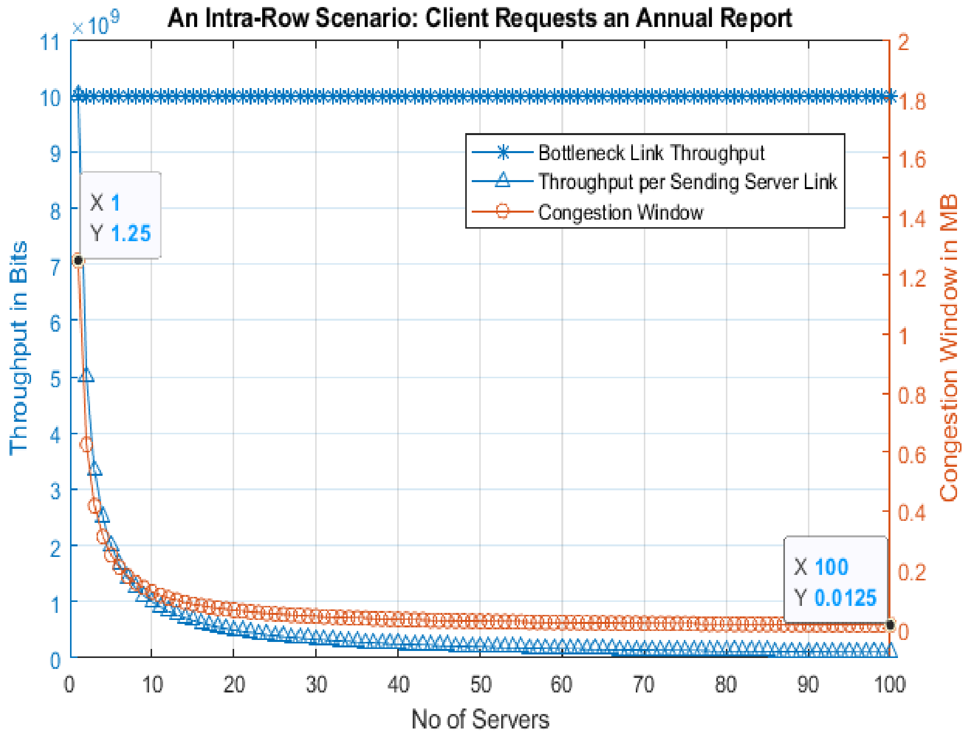


Figure 5.6: Throughput versus numbers of servers when a client requests an annual report in an intra-row arrangement.

5.4.2 An Inter-Row Scenario: Client Server Requests a Movie

In this scenario we supposed a client is requesting a movie that is stored across 150 server. The servers and client are connected to different EOR switches with a 10 Gbps link. let the RTT for this scenario be 1.2 ms. The client's link BDP = 1.2 ms * 10 Gbps, which is equal to 1.5 MB. It is equally divided among all 150 servers and each server gets a 10 KB fair-share, if all 150 servers are transmitting data. Again Figure 5.7 shows a similar behaviour such as the bottleneck link capacity is fully utilized even if the number of sending servers are increasing. Again the client server successfully receives the requested file from multiple storage servers without causing Incast.

5.4.3 Analysis

Similarly, in EOR design, we considered two scenarios in which a client is requesting an annual report and a movie that is stored across multiple servers in an intra-row and

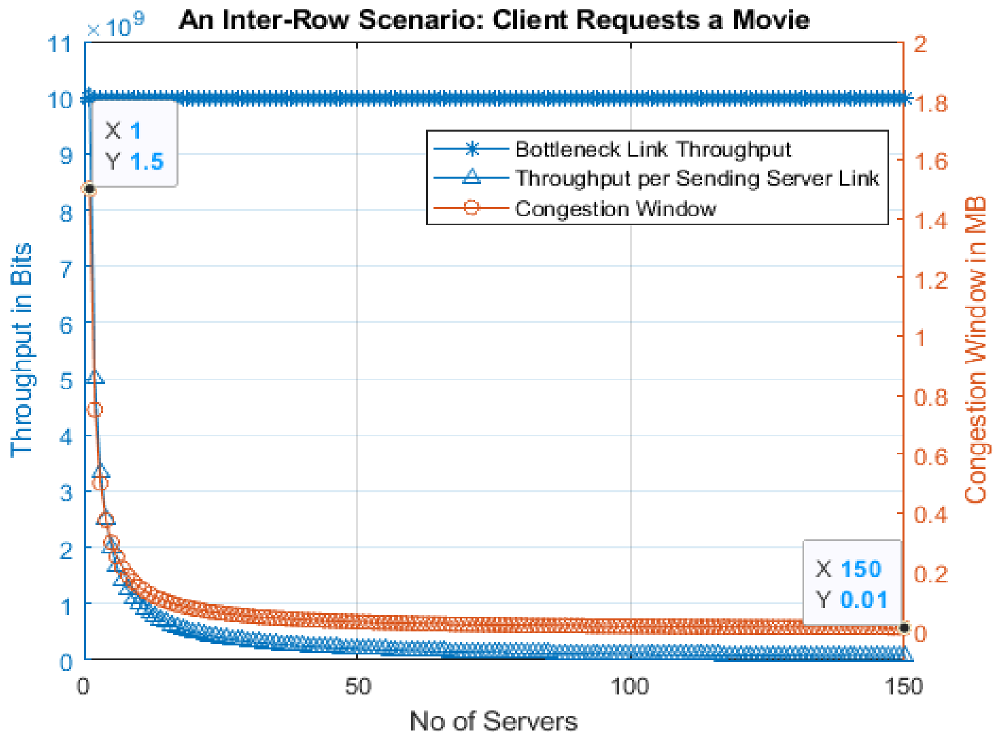


Figure 5.7: Throughput versus numbers of servers when a client requests a movie in an inter-row arrangement.

an inter-row arrangements respectively. It can be observed, when a client requests an annual report, the minimum value of fair-share for a single server is 12.5 KB, which means enough window size is available that can be divided further to increase the number of servers. The maximum value of fair-share is equal to 1.25 MB which is actually same as the client's link BDP. It is still less than the switch buffer capacity. It means that the client's link is the bottleneck. Likewise, in other scenario, when a client requests a movie that is distributed on 150 servers, the minimum value of fair-share is 10 KB, again it shows that number of servers can increase. The maximum fair-share is 1.5 MB that is again less than switch buffer capacity; hence ensuring that switch buffer is not the bottleneck. Figure 5.8 shows the comparison between both scenarios. In this figure, the distribution of BDP is shown via a 3D diagram.

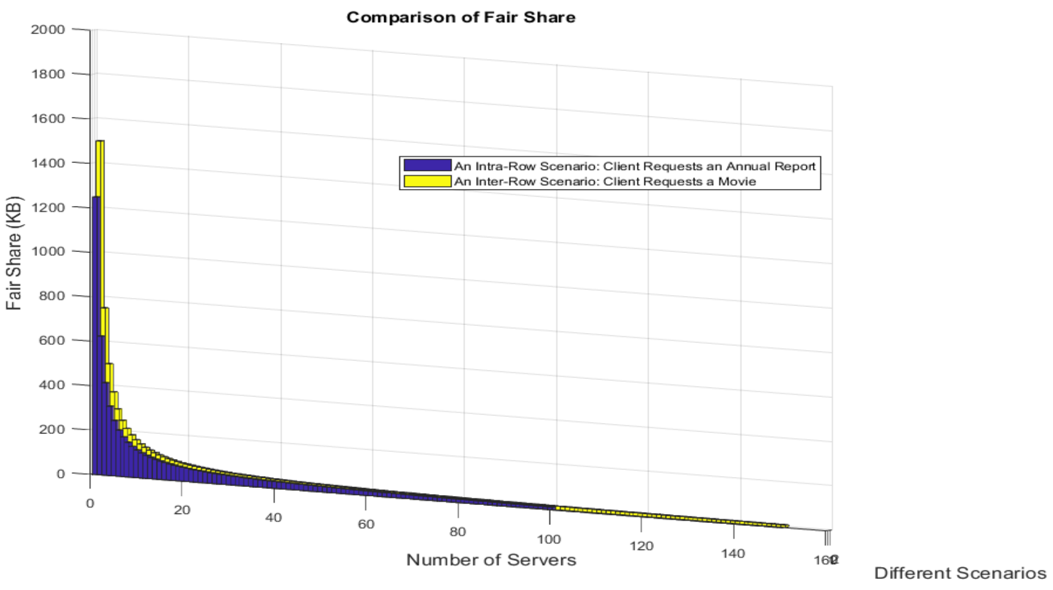


Figure 5.8: Fair-share vs number of servers: Comparison of BDP between different scenarios used in EOR setup.

5.5 Discussion and Future Prospects

M-DyFaShBuP provides fair allocation of buffer capacity for each server. The whole process of M-DyFaShBuP, starting from designing a policy till distributing it among all servers, must be fast enough, else the client's link could not be fully utilized and client would get a lower throughput. The TCP SYN/FIN counter based calculation of the policy makes this process rapid enough to achieve high throughput for the client's link. Table 5.3 show a comprehensive comparison of M-DyFaShBuP with DCTCP [31] and SDTCP [52].

We found that as the number of servers are increased, the congestion window size is also decreased. A problem arises in case of mice flows is that the number of ACK packets are also increased which is a serious problem. In future, we will try to cater this issue. Also we will try to develop a more complex situation considering multiple clients; moderately a many-to-many communication scenario and we will also try to incorporate multiple bottleneck points. Also more research work is required for secure integration of Cloud Computing and IoT.

Table 5.3: Comparison of M-DyFaShBuP with DCTCP and SDTCP

Technique	**Approach**	**Scalability**	**Design**	**Congestion Signal (Use of ECN)**	**Switch Dependency**	**Bandwidth Utilization (Bottleneck Link)**
DCTCP [31]	Distributed	Limited	TOR	Yes	Highly dependent upon threshold set inside switch buffer	Underutilized
SDTCP [52]	Centralized	Limited	TOR	Yes	Continuously measure queue length inside switch buffer	Underutilized
M-DyFaShBuP	Centralized	Unlimited	TOR and EOR	No	Forward a copy of TCP SYN/FIN packets to controller	Fully Utilized

CHAPTER 6

Conclusion

In this thesis document, we have proposed M-DyFaShBuP, a centralized controller-based method for tackling the Incast in data centers. The SDN controller continually monitors the arrival of TCP SYN/FIN packets and formulates a fair-share policy and distributes it among all servers. We have eliminated an extra step required for monitoring the congestion level on the switch. Moreover, M-DyFaShBuP does not require a switch or a receiver for modifying the TCP ACK window field. We have handled the Incast issue for both TOR and EOR designs. Our results verify that M-DyFaShBuP provides full utilization of client's link capacity. It provides high throughput for supporting multiple IoT applications.

References

[1] K. Ashton *et al.*, “That ‘internet of things’ thing,” *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] L. Wang and R. Ranjan, “Processing distributed internet of things data in clouds,” *IEEE Cloud Computing*, vol. 2, no. 1, pp. 76–80, 2015.

[3] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, “Secure integration of iot and cloud computing,” *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[4] T. Dillon, C. Wu, and E. Chang, “Cloud computing: issues and challenges,” in *2010 24th IEEE international conference on advanced information networking and applications*. Ieee, 2010, pp. 27–33.

[5] T. Niemueller, F. Zwilling, G. Lakemeyer, M. Löbach, S. Reuter, S. Jeschke, and A. Ferrein, “Industrial internet of things: Cybermanufacturing systems,” 2016.

[6] B. Shen, N. Chilamkurti, R. Wang, X. Zhou, S. Wang, and W. Ji, “Deadline-aware rate allocation for iot services in data center network,” *Journal of Parallel and Distributed Computing*, vol. 118, pp. 296–306, 2018.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[8] W. Xia, P. Zhao, Y. Wen, and H. Xie, “A survey on data center networking (dcn): Infrastructure and operations,” *IEEE communications surveys & tutorials*, vol. 19, no. 1, pp. 640–656, 2016.

[9] R. S. Couto, M. E. M. Campista, and L. H. M. Costa, “A reliability analysis of datacenter topologies,” in *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 1890–1895.

[10] S. U. Khan and A. Y. Zomaya, *Handbook on data centers*. Springer, 2015.

[11] J. Pang, G. Xu, and X. Fu, “Sdn-based data center networking with collaboration of multipath tcp and segment routing,” *IEEE Access*, vol. 5, pp. 9764–9773, 2017.

[12] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[13] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, “A survey on software-defined networking,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[14] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience with a globally-deployed software defined wan,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[15] H. Farhady, H. Lee, and A. Nakao, “Software-defined networking: A survey,” *Computer Networks*, vol. 81, pp. 79–95, 2015.

[16] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[17] R. Kohavi, R. M. Henne, and D. Sommerfield, “Practical guide to controlled experiments on the web: listen to your customers not to the hippo,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 959–967.

[18] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, “How far can client-only solutions go for mobile browser speed?” in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 31–40.

[19] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” 2008.

[20] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “Dcell: a scalable and fault-tolerant network structure for data centers,” in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 75–86.

[21] V. Vasudevan, A. Phanishayee, and H. Shah, “l. krevat, d. andersen, g. ganger, g. gibson, and b. mueller,” *Safe and effective fine-grained TCP retransmissions for datacenter communication*, 2009.

[22] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu, “Joint virtual machine assignment and traffic engineering for green data center networks,” *ACM SIGMETRICS performance evaluation review*, vol. 41, no. 3, pp. 107–112, 2014.

[23] Y. Ren, Y. Zhao, P. Liu, K. Dou, and J. Li, “A survey on tcp incast in data center networks,” *International Journal of Communication Systems*, vol. 27, no. 8, pp. 1160–1172, 2014.

[24] P. Sreekumari and J.-i. Jung, “Transport protocols for data center networks: a survey of issues, solutions and challenges,” *Photonic Network Communications*, vol. 31, no. 1, pp. 112–128, 2016.

[25] T. Hafeez, N. Ahmed, B. Ahmed, and A. W. Malik, “Detection and mitigation of congestion in sdn enabled data center networks: A survey,” *IEEE Access*, vol. 6, pp. 1730–1740, 2017.

[26] I. Psaras and V. Tsaoussidis, “The tcp minimum rto revisited,” in *International Conference on Research in Networking*. Springer, 2007, pp. 981–991.

[27] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, “Measurement and analysis of tcp throughput collapse in cluster-based storage systems.” in *FAST*, vol. 8, 2008, pp. 1–14.

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

[29] H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, "The panasas activescale storage cluster-delivering scalable high bandwidth storage," in *SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. IEEE, 2004, pp. 53–53.

[30] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 39–64, 2012.

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

[32] Y. A. Bangash, T. Rana, H. Abbas, M. A. Imran, and A. A. Khan, "Incast mitigation in a data center storage cluster through a dynamic fair-share buffer policy," *IEEE Access*, vol. 7, pp. 10 718–10 733, 2019.

[33] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[34] M. Aazam, I. Khan, A. A. Alsaffar, and E.-N. Huh, "Cloud of things: Integrating internet of things and cloud computing and the issues involved," in *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014*. IEEE, 2014, pp. 414–419.

[35] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[36] D. E. O'Leary, "Big data', the 'internet of things'and the 'internet of signs," *Intelligent Systems in Accounting, Finance and Management*, vol. 20, no. 1, pp. 53–65, 2013.

[37] T. Alam, “A reliable communication framework and its use in internet of things (iot),” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, vol. 3, no. 5, pp. 450–456, 2018.

[38] S. H. Mahmud, L. Assan, and R. Islam, “Potentials of internet of things (iot) in malaysian construction industry,” *Annals of Emerging Technologies in Computing (AETiC), Print ISSN*, pp. 2516–0281, 2018.

[39] S. Sudevan and M. Joseph, “Internet of things: Incorporation into healthcare monitoring,” in *2019 4th MEC International Conference on Big Data and Smart City (ICBDSC)*. IEEE, 2019, pp. 1–4.

[40] S. Bera, S. Misra, and A. V. Vasilakos, “Software-defined networking for internet of things: A survey,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[41] Á. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona Lopez, and L. J. García Villalba, “Sdn: Evolution and opportunities in the development iot applications,” *International Journal of Distributed Sensor Networks*, vol. 10, no. 5, p. 735142, 2014.

[42] P. Gonçalves, J. Ferreira, P. Pedreiras, and D. Corujo, “Adapting sdn datacenters to support cloud iiot applications,” in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2015, pp. 1–4.

[43] N. Bizanis and F. A. Kuipers, “Sdn and virtualization solutions for the internet of things: A survey,” *IEEE Access*, vol. 4, pp. 5591–5606, 2016.

[44] W. Chen, P. Cheng, F. Ren, R. Shu, and C. Lin, “Ease the queue oscillation: analysis and enhancement of dctcp,” in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 450–459.

[45] H. Wu, Z. Feng, C. Guo, and Y. Zhang, “Ictcp: Incast congestion control for tcp in data-center networks,” *IEEE/ACM transactions on networking*, vol. 21, no. 2, pp. 345–358, 2012.

[46] G. Wang, Y. Ren, K. Dou, and J. Li, “Idtcp: An effective approach to mitigating the tcp incast problem in data center networks,” *Information Systems Frontiers*, vol. 16, no. 1, pp. 35–44, 2014.

[47] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, “Pac: Taming tcp incast congestion using proactive ack control,” in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 385–396.

[48] C. Lee, Y. Nakagawa, K. Hyoudou, S. Kobayashi, O. Shiraki, and T. Shimizu, “Flow-aware congestion control to improve throughput under tcp incast in datacenter networks,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 3. IEEE, 2015, pp. 155–162.

[49] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, “Rethinking end-to-end congestion control in software-defined networks,” in *Proceedings of the 11th ACM Workshop on Hot Topics in networks*, 2012, pp. 61–66.

[50] S. Jouet, C. Perkins, and D. Pezaros, “Otcp: Sdn-managed congestion control for data center networks,” in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 171–179.

[51] Y. Lu, “Sed: An sdn-based explicit-deadline-aware tcp for cloud data center networks,” *Tsinghua Science and Technology*, vol. 21, no. 5, pp. 491–499, 2016.

[52] Y. Lu and S. Zhu, “Sdn-based tcp congestion control in data center networks,” in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2015, pp. 1–7.

[53] Y. Lu, Z. Ling, S. Zhu, and L. Tang, “Sdtcp: Towards datacenter tcp congestion control with sdn for iot applications,” *Sensors*, vol. 17, no. 1, p. 109, 2017.

[54] J. Xu, H. Guo, J. Wu, J. Lin, D. Zhang, G. Chen, X. Zhang, and C. Chen, “Sig: Solution to tcp incast in sdn network based openflow protocol,” in *Asia Communications and Photonics Conference*. Optical Society of America, 2013, pp. AW4I–5.

[55] A. M. Abdelmoniem and B. Bensaou, “Sdn-based incast congestion control framework for data centers: Implementation and evaluation,” *CSE Dept, HKUST, Tech. Rep. HKUST-CS16-01*, 2016.

[56] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu, “Mitigating incast-tcp congestion in data centers with sdn,” *Annals of Telecommunications*, vol. 73, no. 3-4, pp. 263–277, 2018.

[57] A. M. Abdelmoniem and B. Bensaou, “Reconciling mice and elephants in data center networks,” in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 119–124.

[58] A. A. Akyamac and T. P. Chu, “Non-overlapping rings: A new architecture for designing switch clusters in data centers,” in *2013 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2013, pp. 414–419.

[59] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu, “Sicc: Sdn-based incast congestion control for data centers,” in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[60] J. Gao, A. Oprea, C. Peteanu, M. Polan, A. Trossman, and A. Tsui, “Method and apparatus for deploying and instantiating multiple instances of applications in automated data centers using application deployment template,” Nov. 2 2006, uS Patent App. 11/117,171.