# A Study of the Opportunities and Challenges of Using Edge Computing to Accelerate Cloud Applications

by

Hala Qadi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Hala Qadi 2025

## Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

**Abstract**

I explore the viability of using edge clusters to host latency-sensitive applications and to run services that can improve end-to-end communication performance across both wide area networks (WANs) and 5G environments. The study examines the viability of using edge clusters in three scenarios: accelerating TCP communications through TCP splitting in 5G deployments, hosting an entire application-level service or the latency-sensitive part of an application on an edge cluster, and deploying a TCP splitting service on edge clusters to support WAN communication. I explore these scenarios while varying packet drop rates, communication stacks, congestion control protocols, and TCP buffer sizes.

My findings bring new insights about these deployment scenarios. I show that edge computing, especially through TCP splitting, can significantly improve end-to-end communication performance over the classical communication stack. TCP splitting over the 5G communication stack does not bring any benefit and can reduce throughput. This is because of the unique characteristics of the 5G communication stack. Furthermore, over the classical communication stack, TCP splitting brings higher benefit for flows larger than 64 KB. These findings provide valuable insights into how edge clusters can accelerate TCP communication in different network environments and identify high-impact research ideas for future work.

**Acknowledgements**

I would like to express my sincere gratitude to those who contributed to the completion of this thesis.

First and foremost, I would like to thank my supervisor, Samer Al-Kiswany, for his invaluable guidance, support, and encouragement throughout my research journey. This thesis would never be possible without him.

I am also grateful to Professor Martin Karsten, for his constructive feedback and support, which significantly enhanced the quality of my research.

A special thanks to my friends, Omar and Sara, for their unwavering support, understanding, and encouragement during this journey. Their presence made a significant difference in my experience.

Additionally, I would like to acknowledge my colleagues in the Waterloo's Advanced Systems Lab (WASL), who provided a collaborative and supportive environment.

Lastly, I wish to thank my parents and my lovely sisters for their unconditional love, support, and encouragement throughout this process. I would have never been able to continue my studies without them.

## Dedication

I dedicate this thesis to my loved ones.

# Table of Contents

**Author's Declaration** ii

**Abstract** iii

**Acknowledgements** iv

**Dedication** v

**List of Figures** ix

**List of Tables** xii

**1 Introduction** 1

1.1 Edge as an Accelerator for 5G Communication . . . . . 2

1.2 Deploying Services on MEC . . . . . 4

1.3 Edge as an Accelerator for WAN Communication . . . . . 5

**2 Background** 8

2.1 Edge Computing in 5G . . . . . 10

2.1.1 Multi-access Edge Computing (MEC) . . . . . 10

2.1.2 5G Frame Structure . . . . . 11

2.1.3 Hybrid Automatic Repeat Request (HARQ) . . . . . 11

2.2 Congestion Control . . . 12
2.3 TCP Splitting . . . 13
2.3.1 HAProxy . . . 15

**3 Methodology** **16**
3.1 Verifying Network Emulation Setup . . . 16
3.2 TCP and Buffer Sizes . . . 21
3.3 Choice of Congestion Control Protocols . . . 21
3.4 BBRv1 vs BBRv3 . . . 23

**4 Evaluation** **29**
4.1 Use Case I: Edge as an Accelerator for 5G Communication . . . 29
4.1.1 Impact of Packet Drop Over The Classical Communication Stack . . . 31
4.1.2 Impact of Packet Drop Over The 5G Communication Stack . . . 32
4.1.3 Impact of TCP Buffer sizes over the Classical and 5G Communication Stacks . . . 35
4.1.4 Behavior of Congestion Control Protocols Over the Classical and 5G Communication Stacks . . . 40
4.1.5 Impact of TCP Flow Size Over The Classical Communication Stack 41
4.1.6 Summary . . . 43
4.2 Use Case II: Deploying Services on MEC . . . 44
4.2.1 Impact of Packet Drop Over The Classical Communication Stack . . . 45
4.2.2 Impact of Packet Drop Over The 5G Communication Stack . . . 45
4.2.3 Comparison Between Classical and 5G Communication Stacks . . . 47
4.2.4 Behavior of Congestion Control Protocols Over Classical and 5G Communication Stacks . . . 49
4.2.5 Summary . . . 52
4.3 Use Case III: Edge as an Accelerator for WAN Communication . . . 52
4.3.1 Impact of Proxy Placement . . . 53

4.3.2 Impact of Packet Drop . . . . . 54
4.3.3 Behavior of Congestion Control Protocols . . . . . 56
4.3.4 Impact of TCP Buffer Sizes . . . . . 57
4.3.5 Impact of TCP Flow Size . . . . . 58
4.3.6 Summary . . . . . 59

**5 Related Work** **61**
5.1 Edge Applications . . . . . 61
5.2 Content Delivery Networks . . . . . 61
5.3 TCP Splitting . . . . . 63
5.4 Theoretical TCP Models . . . . . 64
5.5 Congestion Control Surveys . . . . . 65

**6 Conclusion** **67**

**References** **69**

# List of Figures


1.1 Edge as an accelerator for 5G communication. . . . . 3
1.2 Deploying services on MEC. . . . . 5
1.3 Edge as an accelerator for WAN communication. . . . . 6

2.1 A typical edge computing setup. . . . . 9
2.2 TCP Splitting . . . . . 14

3.1 Average link utilization ratio with limited buffers for different link bandwidths at 50 ms RTT. . . . . 22
3.2 Average throughput for different buffer sizes for a 50 ms RTT and 1 Gbps link. . . . . 25
3.3 Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 1 ms RTT link, the error bars show the standard deviation. . 26
3.4 Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 50 ms RTT link, the error bars show the standard deviation. 27
3.5 Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 150 ms RTT link, the error bars show the standard deviation. 28

4.1 Edge as an accelerator for 5G communication. . . . . 30
4.2 Average throughput for different drop rates at 50 ms RTT, 1 Gbps, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation. . . . . 32
4.3 Average throughput for different drop rates at 50 ms RTT, 1 Gbps, and MEC at 5 ms over the classical communication stack. Error bars show the standard deviation. . . . . 33

4.4 Average throughput for different percentages of jittered packets at 50 ms RTT, 1 Gbps, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation. . . . . 34

4.5 Average throughput for different percentages of jittered packets at 50 ms RTT, 1 Gbps, and MEC at 5 ms over the 5G communication stack. Error bars show the standard deviation. . . . . 35

4.6 Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, and MEC at 1 ms. Error bars show the standard deviation. . . . . 36

4.7 Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, 0.5% drop, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation. . . . . 37

4.8 Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, 10 ms jitter affecting 0.5% of packets, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation. . . . . 38

4.9 Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps over different communication stacks. Error bars show the standard deviation. 39

4.10 Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 0.5% drop, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation. . . . . 41

4.11 Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 10 ms jitter affecting 0.5% packets, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation. . . . . 42

4.12 Average response time for different flow sizes at 50 ms RTT, 1 Gbps, and 0.5% drop over the classical communication stack. . . . . 43

4.13 Deploying services on MEC. . . . . 44

4.14 Average throughput for different drop rates at 1 ms RTT and 1 Gbps over the classical communication stack. Error bars show the standard deviation. 46

4.15 Average throughput for different drop rates at 5 ms RTT and 1 Gbps over the classical communication stack. Error bars show the standard deviation. 47

4.16 Average throughput for different percentages of jittered packets at 1 ms RTT and 1 Gbps over the 5G communication stack. Error bars show the standard deviation. . . . . 48

4.17 Average throughput for different percentages of jittered packets at 5 ms RTT and 1 Gbps over the 5G communication stack. Error bars show the standard deviation. . . . . 49

4.18 Average throughput for different congestion control protocols at 0.5% drop, 1 ms RTT, and 1 Gbps over the classical communication stack. Error bars show the standard deviation. . . . . 50

4.19 Average throughput for different congestion control protocols at 10ms jitter affecting 0.5% of packets, 1 ms RTT, and 1 Gbps over the 5G communication stack. Error bars show the standard deviation. . . . . 51

4.20 Edge as an accelerator for WAN communication. . . . . 53

4.21 Average throughput for different proxy placements at 150 ms RTT, 1 Gbps, and 0.5% packet drop. Error bars show the standard deviation. . . . . 54

4.22 Average throughput for different drop rates at 150 ms RTT, 1 Gbps, and edge at 75 ms. Error bars show the standard deviation. The smaller figure in the top right corner provides a zoomed-in view of the lower bars. . . . . 55

4.23 Average throughput for different congestion control protocols at 150 ms RTT, 1 Gbps, 0.5% drop, and edge at 75 ms. Error bars show the standard deviation. . . . . 56

4.24 Average throughput for different buffer configurations at 150 ms RTT, 1 Gbps, and edge at 75 ms. Error bars show the standard deviation. . . . . 57

4.25 Average throughput for different buffer configurations at 150 ms RTT, 1 Gbps, 0.5% packet drop, and edge at 75 ms. Error bars show the standard deviation. . . . . 59

4.26 Average response time for different flow sizes at 150 ms RTT, 1 Gbps, and 0.5% drop. . . . . 60

# List of Tables


3.1 Verifying delay configuration with Linux TC, standard deviation is less that 0.1%. . . . . 17

3.2 Verifying rate configuration with Linux TC, standard deviation is less than 1.1%. . . . . 18

3.3 Verifying drop rate configuration on UDP traffic with Linux TC, standard deviation is less than 5%. . . . . 18

3.4 Verifying Drop Rate Configuration on TCP traffic with Linux TC, standard deviation is less than 2%. . . . . 19

3.5 Verifying percentage of 5 ms jittered packets, standard deviation is less than 4%. . . . . 20

3.6 Verifying percentage of 10 ms jittered packets, standard deviation is less than 5%. . . . . 20

# Chapter 1

# Introduction

Edge computing is rapidly gaining popularity and is deployed widely in today's connected world. In this day and age, the limitations of conventional cloud computing models have become more pronounced as more applications demand low-latency communications and real-time processing. Edge computing brings computational power closer to clients, thereby reducing latency and allowing for faster processing. This transition to decentralized edge nodes enables more efficient and responsive applications, especially in areas like smart cities, autonomous vehicles, and the Internet of Things (IoT) [59].

Edge computing is emerging as a crucial component in two domains: wide area networks (WANs) and the fifth generation mobile network (5G), each with its specific setup and purposes. In WANs, edge nodes are positioned at strategic locations, normally close to major gateways or at the edges of data centers. By processing data closer to its source, edge computing in WANs serves the purpose of reducing the amount of data that traverses the network. This reduces bandwidth congestion as well as response times for latency-sensitive applications.

In 5G networks, edge computing is embedded in the mobile network infrastructure, and is typically located near the radio access network. This proximity enables ultra-low latency processing which is essential for applications such as virtual reality (VR) and augmented reality (AR). The purpose of edge computing in 5G is to take advantage of the upgraded capabilities of the network, including higher data rates and more reliable connections, as data is being processed as close to the user as possible.

In the context of WANs, the unique deployment scenario of edge clusters on high latency network paths makes them well-suited for deploying TCP acceleration techniques, such as

TCP splitting. The strategic placement of edge clusters mitigates the impact of latency, consequently improving the performance of TCP connections across the network.

TCP splitting is a term that refers to splitting a TCP connection into two or more sub-connections using TCP proxies. When a long-distance connection is split into two or more connections, each sub-connection has a smaller Round-Trip Time (RTT). It reduces the effective RTT as the feedback-loop of the TCP connection is reduced, which enhances the end-to-end transfer throughput. TCP proxies play a huge role as they buffer packets, which enables the dropped packets to be retransmitted from the proxy (the shorter connection) rather than from the server. It is essential to note that TCP proxies enhance the throughput of data transfer without the need to modify the TCP/IP protocol stack of the end-hosts [85]. This is particularly useful for long-distance connections, as they suffer from low throughput because of long delays and high packet loss. As edge clusters shorten the distance between the users and where data are processed, they make TCP splitting notably effective. The close proximity allows TCP proxies to handle retransmissions quickly, thereby reducing latency and optimizing throughput.

Previous research efforts show that TCP splitting can enhance communication performance for satellite communications [50] over a classical communication stack, which typically uses TCP over Ethernet. It is not obvious whether TCP splitting will bring benefit to new setups in 5G environments. This is because 5G uses a different communication stack that utilizes the HARQ protocol at the link layer to provide reliable data transmission. The fifth generation mobile network also uses a specialized frame structure (Section 2.1.2). This combination causes the 5G communication stack to have fundamentally different reliability and latency properties compared to the classical communication stack. Moreover, the latency division between the sub-connections is different than the ones that have been explored in inter-terrestrial communications.

In this thesis, I seek to explore the viability of using edge clusters to improve end-to-end communication performance across both WANs and 5G environments. The study focuses on accelerating TCP communications through the use of TCP splitting and edge clusters. To address this, I study three use cases: edge as an accelerator for 5G communication, deploying services on Multi-access Edge Computing (MEC), and edge as an accelerator for WAN communication.

## 1.1 Edge as an Accelerator for 5G Communication

This use case explores the viability of accelerating end-to-end TCP throughput through using TCP splitting. It is challenging to achieve high TCP performance over 5G deploy-

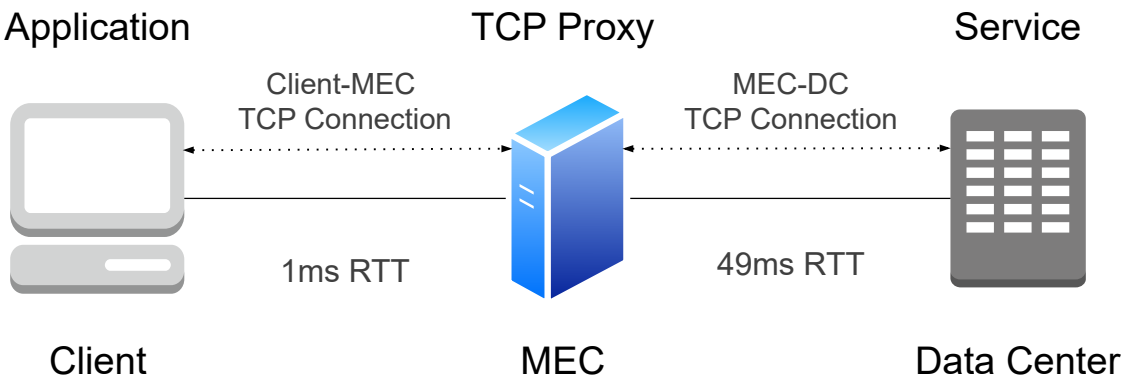



Figure 1.1: Edge as an accelerator for 5G communication.

ments. This is because of the heterogeneity of the deployment. The network has two distinct segments with widely different characteristics. On the radio access network, the latency is small, in the range of 1-5 ms, but drop or error rates are high. In contrast, core networks have higher reliability but significantly longer delay, in the range of 40 - 200 ms. The recovery time of most congestion control protocols, especially loss-based protocols, is related to the latency. With high loss rates and latency, recovery times increase, significantly reducing throughput. Splitting the connection into two parts, a connection in the radio access network (between the client and the MEC), and a connection between the MEC and the cloud data center, has the potential to improve TCP performance. The long-distance connection will not suffer from the high loss in the radio access network, while the short radio connection will recover faster. This approach can reduce latency and significantly improve end-to-end throughput.

Figure 1.1 shows a possible deployment scenario. In this case, the data center hosts the service, and the MEC that is close to the user acts as a TCP proxy. When the TCP proxy receives a connection request, it creates a separate TCP connection to the server at the data center. The MEC buffers packets sent on the client-to-MEC connection, and sends them over the MEC-to-server connection. The latency between the end user device and the MEC can be in the range of 1ms to 5ms, while the latency between the MEC and the data center can be 50ms or more.

Understanding how various network parameters affect the overall performance is essential for communication optimization, especially in 5G environments. To explore the benefits of TCP splitting in this context, I address the following key questions:

- What is the impact of packet drop on end-to-end throughput over the classical communication stack while varying the MEC placement?
- What is the impact of packet drop on end-to-end throughput over the 5G communication stack?
- What is the impact of TCP buffer sizes on end-to-end throughput? Are the default buffer values enough?
- How do different congestion control protocols perform over both the classical and 5G communication stacks?
- Under what traffic conditions does TCP splitting enhance the performance? For instance, at what flow size does TCP splitting bring performance benefits?

I examine this topic further in Section 4.1.

## 1.2 Deploying Services on MEC

Deploying services directly on the MEC can enhance application performance by bringing computing resources closer to the user and significantly reducing communication latency. Figure 1.2 shows a possible deployment. In this scenario, the service is either completely hosted at the edge, or the latency-sensitive portion of it is hosted at the edge. The service is deployed on the 5G MEC that is close to the end user device. The latency between the end user device and the MEC can be in the range of 1 ms to 5 ms. The edge service may still access the central cloud infrequently for monitoring, billing, and archival purposes.

The implications of deploying services on MEC servers close to the user are not well-understood. While deploying services with 1 ms latency is the norm in data centers using the classical TCP/IP over Ethernet stack, deploying services with 1 ms latency on the 5G communication stack has not been studied. This is because 5G uses a significantly different communication stack with different failure semantics and latency characteristics. The fifth generation networks use a special communication frame structure [1], and the Hybrid Automatic Repeat Request (HARQ) protocol for error control at the link layer [19]. HARQ is a reliable link layer protocol that, when used on top of 5G frames, can introduce latency spikes of up to 10 ms - ten times the average RTT. This increase in latency may cause timeouts and trigger congestion control mechanisms. Notably, this is not an issue in typical 5G deployments over long distances, where adding 10 ms latency to

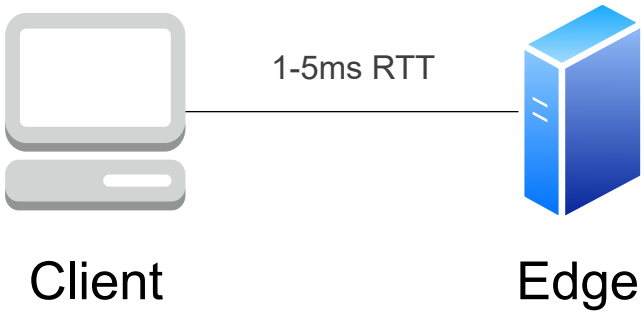



Figure 1.2: Deploying services on MEC.

a 50 ms RTT will not cause TCP timeouts. However, when the RTT is 1 ms, adding 10 ms latency can cause TCP timeouts and reduce throughput.

In this thesis, I aim to study the performance implications of running services on the MEC. Specifically, I aim to explore the following questions:

- What is the impact of packet drop on end-to-end throughput over the classical communication stack?
- What is the impact of packet drop on end-to-end throughput over the 5G communication stack?
- How do different congestion control protocols perform over both the classical and 5G communication stacks?

Section 4.2 provides a detailed analysis of these questions.

## 1.3 Edge as an Accelerator for WAN Communication

Edge clusters can accelerate TCP communications over wide-area networks through TCP splitting. With long-distance TCP connections, even a small packet drop can substantially affect the performance, as these links require extended recovery time. TCP splitting can alleviate this problem by splitting the long-distance connection into two shorter sub-connections. This allows for faster recovery rates in each sub-connection, which reduces the impact of packet drop and improves the end-to-end throughput.

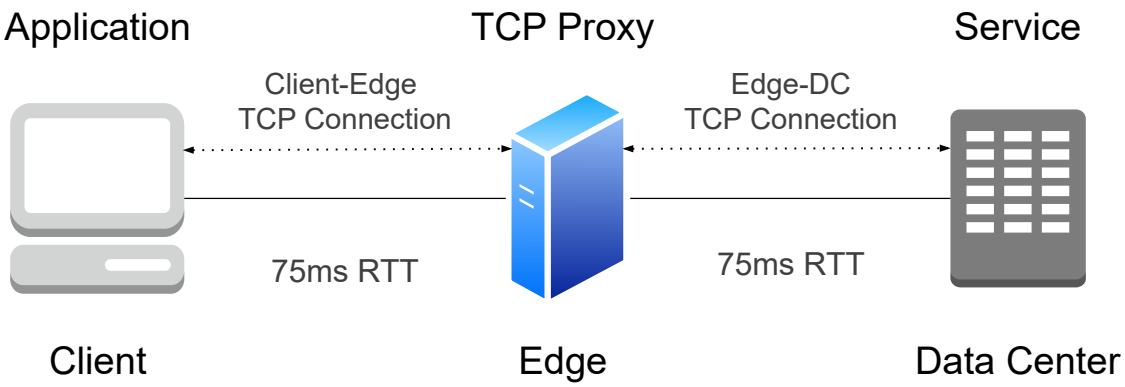



Figure 1.3: Edge as an accelerator for WAN communication.

Figure 1.3 presents a possible deployment. The data center hosts the service, and the edge is in the middle between the client and the server. When a connection request reaches the edge, it creates another TCP connection to the server. This way, the edge maintains two separate TCP connections for the client. The edge buffers packets received from the client-to-edge connection and transmits them over the edge-to-server connection. The end-to-end latency between the data center and the end user can be in the range of 40 ms to 200 ms. With the edge splitting the connection, each sub-connection will have lower RTT (half if the edge is in the middle).

In this thesis, I assess the viability of using edge clusters for TCP acceleration through TCP splitting. Specifically, I aim to address the following questions:

- What is the impact of TCP proxy placement on end-to-end throughput?
- What is the impact of packet drop on end-to-end throughput?
- How do different congestion control protocols perform?
- What is the impact of TCP buffer sizes on end-to-end throughput? Are the default buffer values enough?
- Under what traffic conditions does TCP splitting enhance the performance? For instance, at what flow size does TCP splitting bring performance benefits?

Section 4.3 studies these questions in detail.

To answer these questions, I follow an empirical approach using a cluster of three nodes with network emulation to mimic the aforementioned use cases. I validate the emulation setup, configure TCP buffer sizes, and select representative congestion control protocols. I then run experiments to address the aforementioned questions.

The results show that TCP splitting brings benefit to most cases over the classical communication stack. However, TCP splitting over the 5G communication stack does not bring any benefit and can reduce throughput. This is because the 5G communication stack causes latency spikes as it hides packet drop from the TCP layer. With the edge being close to the client, this latency spike is significant compared to the link RTT, resulting in reducing the congestion window and reducing throughput. Moreover, over the classical communication stack, TCP splitting brings higher benefit for flows larger than 64 KB.

The rest of this thesis is organized as follows. Chapter 2 provides the necessary background. Chapter 3 details the methodology and experimental setup. Chapter 4. presents the evaluation of the use cases. Chapter 5 discusses related work. Finally, I conclude in Chapter 6.

# Chapter 2

# Background

Edge computing is a paradigm in which clusters of computing and storage resources are distributed closer to clients. This approach reduces the reliance on centralized data centers, thereby decentralizing data processing. This minimizes latency and improves real-time data processing, which makes it essential for latency-sensitive applications such as healthcare monitoring systems. Figure 2.1 shows the interaction between a client, an edge node, and a data center in a typical edge computing setup.

Edge computing introduces multiple benefits that include:

- Ultra-low Latency: Processing data at the edge reduces latency. This is highly beneficial for applications that require real-time decision-making, such as healthcare and traffic management systems.
- Higher reliability: Applications can still function if central data centers are down or if the WAN is down.
- Improved Privacy: When using private MEC, the risk of data exposure is reduced when sensitive data is processed closer to the source [11].
- Enhanced Scalability. Edge computing offers a new opportunity for scaling a service by scaling at the edge instead of scaling at the data centers.

The concept of edge computing is as old as the WAN itself. It initially manifested through the deployment of web proxy servers, web caches, and content distribution networks [69]. The focus of these scenarios is on reducing the latency for serving static web

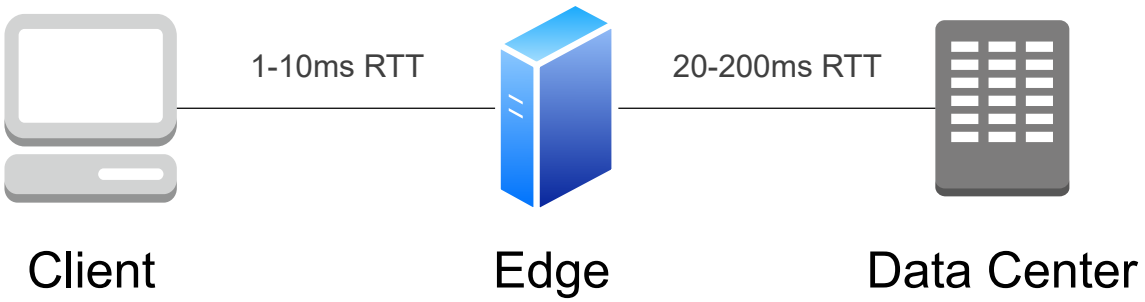



Figure 2.1: A typical edge computing setup.

content to clients and reducing the stress on a handful of globally distributed data centers. Edge computing for CDN is common today, with major providers such as Cloudflare [16] and Akamai [5].

Recent offerings of edge clusters expand the offered services beyond content delivery. Cloud providers offer edge clusters as extensions to cloud computing services offered by data centers. This includes computing and storage services in addition to Software as a Service (SaaS) offerings. For instance, Amazon Outpost [71] and Amazon Wavelength [72] provide Amazon Web Services (AWS) infrastructure to any edge or on-premises location. Moreover, Azure Stack Edge [58] provides the ability to run containerized applications and virtual machines at the edge to process and filter data locally, then send only the filtered results to the cloud for storage or further processing.

Recent research efforts explore how to leverage the proliferation of edge resources to improve a wide set of applications. For instance, recent efforts advocate edge computing for Internet of Things (IoT) applications that utilize sensor streams to enhance interactive applications [63]. Their results show that edge computing is essential for latency-sensitive applications including virtual and augmented reality. Hayat et al. [29] evaluate edge computing in 5G for autonomous drone navigation. Huang et al. [35] promote edge computing into multirobot simultaneous localization and mapping (SLAM). Furthermore, Bing et al. [10] propose an edge computing based processing architecture for coal mining systems.

Edge computing plays a critical role in mobile communication. The 5G network significantly reduces latency and improves throughput. However, with these advances, the tens of milliseconds latency to the nearest data center is not acceptable. To address this challenge, 5G includes edge clusters, known as MEC. The MEC can host complete applications or the latency-sensitive parts of an application, bringing computation closer to the

client and lower service latency. Recent efforts explore using MEC for smart manufacturing, automated guided vehicles, and auto visual inspection [60], as well as intelligent video acceleration, video analytics, augmented reality, and connected vehicles [31].

My observation is that edge clusters are uniquely positioned on the communication path between end-users and the central data center. Consequently, because of this strategic placement, I will be exploring MECs for accelerating TCP communication.

In this section, I cover the background relevant to this use case, including the 5G frame structure, link layer error correction protocols, congestion control mechanisms, and TCP splitting.

## 2.1 Edge Computing in 5G

The fifth generation communication [3] provides low latency vital for latency-sensitive applications. The MEC improves 5G by processing data closer to its source, thereby enabling real-time responses.

5G utilizes the HARQ protocol at the link layer to provide reliable data transmission, alongside a specialized frame structure. At the transport layer (layer four of the OSI model), it uses well-known transport protocols such as TCP. I refer to this combination as the 5G communication stack. In contrast, traditional communication systems typically use TCP over Ethernet, which I call the classical communication stack.

### 2.1.1 Multi-access Edge Computing (MEC)

MEC is a specific type of edge computing that focuses on integrating the capabilities of cloud computing at the edge of the network. Although MEC is often associated with cellular networks, specifically in the context of 5G, it can be deployed in several network environments like Wi-Fi, enterprise, and broadband networks. It was initially called Mobile Edge Computing. However, the name later evolved to Multi-access as it supports different types of network access technologies, beyond just cellular networks.

By processing data closer to its source, MEC enables high-bandwidth and low-latency applications. This makes MEC particularly useful for applications such as augmented reality, autonomous driving and Internet of Things (IoT) applications [74]. MEC delivers various advantages such as reduced latency, better performance, enhanced security, and optimized bandwidth.

In the following, I present the background needed to understand the design choices in 5G that impact communication throughput and latency. I start by discussing the low level 5G frame structure, link layer protocol, then congestion control protocols.

### 2.1.2 5G Frame Structure

The fifth generation communication employs a novel frame structure that profoundly changes how packet transmission is handled [1]. This new structure has significant impact on latency and throughput, directly affecting the performance of different transport protocols. In this section, I focus my discussion on the relevant aspects of the frame structure that impact the communication performance.

In 5G, data between the source and the destination is transmitted over radio frames. The length of each radio frame is fixed to 10 ms and is divided into 10 subframes, each 1 ms long. Each subframe is further divided into several slots in the time domain. 5G uses Orthogonal Frequency Division Multiplexing (OFDM). In an OFDM system, there are several parameters that define the physical layer structure. These include subcarrier spacing and symbol duration. Generally, the slot gets shorter as the subcarrier spacing gets wider. This set of parameters is often known as the Numerology.

Numerology affects the number of slots in a subframe. Within each slot, there are resource blocks, which are allocations of frequency and time resources. Clients, normally referred to as User Equipment (UE), are assigned specific resource blocks for data transmission. The UE resource allocation is highly dynamic; a UE could be assigned resource blocks in one or more slots in a frame or in multiple frames.

If data is lost, the UE can retransmit the lost data in the next slot allocated to it. This means that the UE must wait until new resource blocks are allocated, depending on available resources. Hence, the time that the UE would need to wait varies [4].

### 2.1.3 Hybrid Automatic Repeat Request (HARQ)

HARQ [19] is an error correction protocol used within the link layer in modern 5G communications [2, 20]. HARQ is designed for communications over channels with high packet error rates such as wireless communication. HARQ is different from the Ethernet, the default Medium Access Control (MAC) protocol in data centers. Typically, standard MAC protocols focus on efficient channel access and collision avoidance. HARQ enhances the

reliability of data transmission by combining forward error correction (FEC) and automatic repeat request (ARQ). Before transmission, packets are encoded using FEC with redundant information. When a packet is received, error checks are performed using the redundant information. If errors are found, the receiver sends a negative acknowledgement to trigger a retransmission. Nevertheless, the erroneous packet is not discarded; it gets combined with the retransmitted packet(s) to enhance error correction capability. Moreover, HARQ uses timers to retransmit lost data in case of timeouts. This makes HARQ particularly useful for noisy and variable wireless channels, which are not a concern in the stable environments within data centers.

HARQ offers a reliable link layer communication. It handles packet loss and corruptions. A packet loss or corruption that can not be addressed by FEC requires retrasmitting the data in the next available slot for that UE. Consequently, data loss or corruption manifests as communication jitter to upper transport protocols. The impact of this fundamental difference on congestion protocols operations has not been studied before.

## 2.2 Congestion Control

Congestion control aims to regulate data traffic to prevent network congestion which can lead to performance degradation and network instability. It involves protocols that dynamically modify transmission rates based on network conditions.

Congestion control protocols dynamically adjust the sender's transmission rate based on real-time feedback from the network. While congestion typically causes packet delays and drops, these protocols differ in the way they infer and react to loss. Congestion control techniques can be classified into four main categories: delay-based, loss-based, hybrid, and Explicit Congestion Notification (ECN) based protocols.

- Loss-based congestion control protocols use packet loss as a signal to modify transmission rates. Notable examples include Reno (standard TCP), CUBIC [27], and Westwood [55]. CUBIC modifies the traditional TCP window growth function to a cubic form, making it independent of RTT. During steady state, the window is increased aggressively when far from saturation, and more cautiously when nearing it. TCP Westwood estimates end-to-end bandwidth to distinguish between losses caused by congestion and those due to poor radio conditions.

- Delay-based congestion control protocols, such as TCP Low Priority (TCP-LP) [44] and TCP BBR [13], utilize packet delays to indicate congestion. TCP-LP, a dis-

tributed algorithm, aims to use only the excess bandwidth of the network by leveraging one-way packet delays as congestion signals. Its congestion avoidance policy is TCP-transparent, meaning that it takes advantage of the bandwidth that TCP flows leave unused, in a non-intrusive way. TCP BBR employes Kleinrock's [40] formal model of congestion and its optimal operating point. It estimates the network path's maximum bandwidth and measures the minimum round-trip time to determine the transmission rate that avoids congestion.

- Hybrid protocols like TCP-Illinois [47] combine both delay and loss as congestion signals. TCP-Illinois uses loss to trigger window changes, and delay to specify the amount of the adjustment, creating a concave curve for the window size.

- Protocols that use Explicit Congestion Notification (ECN) for early congestion detection. These protocols require support from routers in the network. When a router starts filling its buffers, it sends explicit messages to notify the senders to reduce their transmission rates. For example, DATA Center TCP (DCTCP) [7] leverages ECN signals to deliver multi-bit feedback to the end-hosts. It responds proportionally to the degree of congestion, significantly enhancing performance in data centers. This behavior is primarily triggered by buffer overflow. In ECN-based protocols, the handling of radio drops depends on their design, classifying them as either loss-based or delay-based.

## 2.3 TCP Splitting

TCP splitting is a technique used to accelerate wide area communication. It is used to enhance the throughput of data transfers. As the name indicates, TCP splitting splits a TCP connection into two or more smaller connections by means of a TCP proxy as shown in Figure 2.2. Each sub-connection is managed separately, which results in latency reductions and throughput enhancements. TCP splitting implementations are available in many open-source software. They offer robust and accessible solutions for several applications. Two well-known examples of such software are HAProxy [79] and Nginx [77].

TCP splitting typically improves recovery time in case of packet loss. TCP employs congestion control techniques to control the transmission rate and avoid congestion episodes. This is achieved by adjusting the congestion window (cwnd) based on network conditions. The congestion window represents the maximum number of packets that can be transmitted without an acknowledgement.

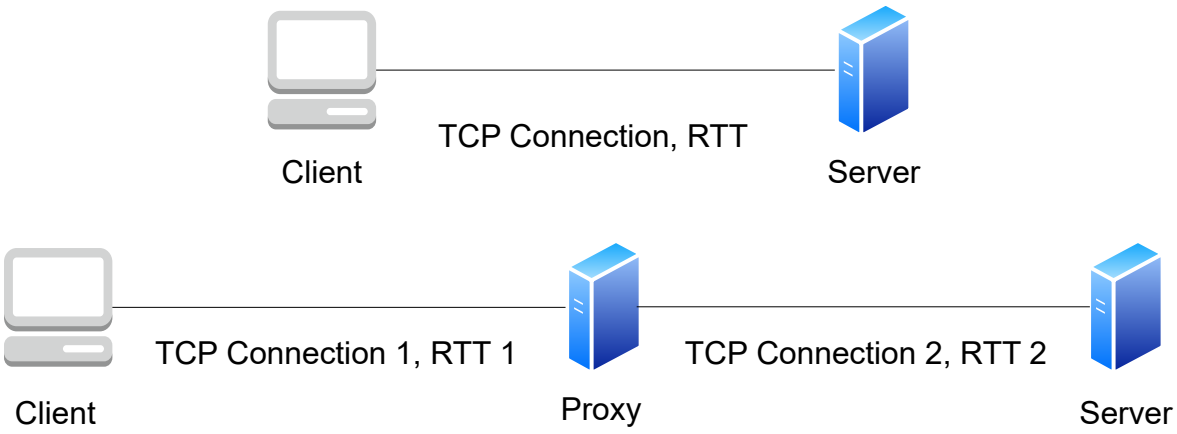



Figure 2.2: TCP Splitting

For instance, loss-based congestion control protocols, including standard TCP Reno, react to packet loss as a signal of congestion. When a loss occurs, the cwnd is significantly reduced, typically to half of its previous value [8]. As more packets are acknowledged, the window increases, which occurs approximately every round-trip time (RTT). In standard TCP, it takes $W/2 * RTT$ seconds for the cwnd to recover to its size before the congestion event, where W is the cwnd at the time of the loss. This can result in under-utilizing the network and reduces throughput, especially in high bandwidth and high RTT networks. While other protocols may not follow this exact formula, they are still impacted by RTT.

This is where TCP splitting can improve throughput. By splitting the long TCP connection into multiple shorter connections, the effective RTT is reduced. Consequently, the feedback-loop of the TCP connection is shortened, enhancing end-to-end transfer throughput without modifying the TCP/IP protocol stack of the end-to-end hosts [85]. Furthermore, since the TCP rate is inversely proportional to RTT, and the window size is directly related to throughput, reducing RTT increases the window size, which results in increasing the throughput. This is shown by the Mathis model [57] for standard TCP Reno:

$$Throughput = \frac{MSS * C}{RTT * \sqrt{p}}$$

Where MSS is the maximum segment size, C is a constant, RTT is the round-trip time, and p is the packet loss rate.

### 2.3.1 HAProxy

HAProxy is a proxy that is reliable, free, and very fast. It offers several features such as load balancing, high availability and proxying for both HTTP and TCP-based applications. HAProxy is quite suitable for high-traffic websites, and has become the standard open source load balancer. Moreover, it is included in most of the mainstream Linux distributions, and is frequently deployed in cloud platforms [79]. Most of its processing is handled by the kernel. HAProxy uses Linux TCP `slices` tool to forward data between two sockets, thereby avoiding memory copies [80].

Nginx [77] is another software that operates as an HTTP and a proxy server. It also can be configured to perform TCP splitting. I choose to use HAProxy because it directly supports TCP splitting and is more popular as a TCP proxy.

# Chapter 3

# Methodology

In this section, I explain the experimental setup as well as the setup verification. In the experiments, I use three nodes on CloudLab [17] to build a setup similar to the one depicted in Figure 2.2. Each machine has an Intel(R) Xeon(R) CPU E5-2450 at 2.10GHz single-socket processor with 16 cores, 15 GB RAM, and 10 Gbps NIC. The nodes are running Ubuntu 20. Unless otherwise specified, I use TCP CUBIC [27], the Linux default TCP congestion control. I configure TCP with 2 GB buffers to eliminate any impact that buffer limitations might have on throughput. I use Linux Traffic Control (TC) [45] to emulate different network conditions, and iperf3 [36] to generate load and measure end-to-end throughput. I configure HAproxy [79] as a TCP proxy. Unless stated otherwise, I use 3 nodes: one node to run the client iperf, one for the server iperf, and one to run the TCP proxy.

The rest of this chapter details my efforts to verify the accuracy of my setup, including verifying network emulation, choosing TCP buffer sizes, and selecting representative congestion control protocols.

## 3.1 Verifying Network Emulation Setup

Before running any experiments, I verify that the deployment can accurately emulate different network conditions in terms of network delay, bandwidth, packet loss rates, and jitter, as these are the configuration parameters in the experiments. To emulate different network conditions, I use Linux Traffic Control (TC) to control the traffic at the end hosts. TC controls outbound traffic on both the client and server sides, while inbound traffic remains uncontrolled.

| Configured Delay (ms) | Expected Delay (ms) | Measured Delay (ms) | Error Rate |
|---|---|---|---|
| 10 | 10.225 | 10.3 | 0.7% |
| 50 | 50.225 | 50.3 | 0.1% |
| 100 | 100.225 | 100.3 | 0.1% |

Table 3.1: Verifying delay configuration with Linux TC, standard deviation is less that 0.1%.

**Network delay emulation**. To verify the added delay, I configure two nodes, node1 and node2. The configuration of each node's interface adds a certain amount of delay. The total RTT between node1 and node2 is the summation of the delays on the two nodes and network communication overhead. I run an experiment between the two nodes with added delays of 10 ms, 50 ms and 100 ms. I measure the delay using the ping utility with a count of 50 ECHO_REQUEST packets.

Table 3.1 shows the average of three runs of this experiment, and the error rate compared to expected delay. The results show that emulation of network delays is accurate. The difference between the set delay and the actual measured delay is not noticeable, and the error rate is less that 1% in all cases. The additional 0.225 ms in the expected delay column is the baseline ping measurement without any added delay. The error rate increases as the configured delay decreases, since the propagation delay contributes proportionally more to the overall overhead.

**Bandwidth Emulation**. To verify the bandwidth configuration, I run an experiment in which I configure each node's interface with the following rates: 100 Mbps, 1 Gbps and 5 Gbps. I also include the results of running the experiment without any configuration as a baseline, that is the default 10 Gbps of the interfaces. Unless otherwise specified, each experiment runs for one minute using iperf3.

Table 3.2 shows the average of three runs and the error rate compared to the configured bandwidth. For the default 10 Gbps configuration, the results demonstrate that iperf3 is able to saturate the link. Its performance is only 6% lower than the ideal throughput of 10 Gbps. This supports the decision to use iperf3 with TCP to verify rate configuration. The results demonstrate that bandwidth emulation is accurate. The findings indicate minor discrepancies between the configured bandwidth and the measured throughput, with error rates between 3.9 - 4.4%. The error rates increase as the configured bandwidth increases, but the difference is not significant.

**Packet Drop Emulation**. To verify the added drop rates, I configure the interfaces

| Configured Bandwidth | Measured Throughput Using iperf3 (mbps) | Error Rate |
|---|---|---|
| 100mbps | 96.1 | 3.9% |
| 1gbps | 957 | 4.3% |
| 5gbps | 4780 | 4.4% |
| No TC (10gbps interface) | 9390 | 6.1% |

Table 3.2: Verifying rate configuration with Linux TC, standard deviation is less than 1.1%.

| Expected Drop Rate | Custom Loss Measurement | Custom Loss Measurement Error Rate | iperf3 | iperf3 Error Rate |
|---|---|---|---|---|
| 1% | 0.996% | 0.4% | 1.08% | 8% |

Table 3.3: Verifying drop rate configuration on UDP traffic with Linux TC, standard deviation is less than 5%.

of both nodes with an added drop rate of 1%. I test the configuration with running iperf3 for one minute, and then average five runs. I verify the loss rate using TCP and UDP tests. The reason for using UDP is that TCP hides packet loss from the application layer, while UDP makes it possible to detect lost packets.

I use two implementations to measure the loss rate in UDP. First, I use iperf3 with a datagram size of 1 KB. iperf3 reports the loss rate. Second, I develop a custom UDP loss measurement script that sends a million 1 KB datagrams and counts the number of packets that reach the destination.

Table 3.3 presents the loss rate measured with the custom UDP loss measurement script and iperf3. The measured packet loss rate closely matches the configured loss rate, with an error rate of 0.4% when using the custom measurement script and 8% when measured by iperf3.

Since the UDP tests are unidirectional, I take drop rate verification one step further by measuring drop rates with TCP tests. Packet drop causes TCP to retransmit the dropped packet. I calculate the drop rate by tracking the number of retransmissions, and dividing it by the total number of sent segments. There are three configuration scenarios on the two nodes, 1% on the sender's interface only, 1% on the receiver's interface only, and 1% on the interfaces of both nodes.

Table 3.4 shows the measured loss rate in each scenario. In the first and last scenarios,

| Configured Drop Rate | Expected Drop Rate | Measured Drop Rate | Error Rate |
|---|---|---|---|
| 1% on client side | 1% | 1.03% | 3% |
| 1% on server side | 0% | 0% | 0% |
| 1% on both sides | 1% | 1.02% | 2% |

Table 3.4: Verifying Drop Rate Configuration on TCP traffic with Linux TC, standard deviation is less than 2%.

the measured packet drop rate matches the configured loss rate, with error rates of only 3% and 2%, respectively. However, the measured loss rate in the second scenario is 0%. This is expected as losing segments from the receiver to the sender (acknowledgments) is not as significant as losing data segments. TCP acknowledgments are cumulative. When a TCP acknowledgement (ACK) packet is lost, it is implicitly acknowledged when the next acknowledgment is received.

The minor discrepancies between the measured and configured loss rates in both the UDP and TCP tests are acceptable, as TC's default loss configuration follows a random model. Given that this thesis focuses on data being sent from clients to servers (upload data), I select the third scenario, drop rate on the client and server interfaces, for configuring the loss rate in the experiments.

**Verification of HARQ and 5G Scheduling**. HARQ aims to provide a reliable link layer protocol through retransmitting lost packets. Consequently, when a packet is dropped or corrupted it is retransmitted, leading to higher jitter. As I could not find any open-source implementations of HARQ or 5G frame and scheduling algorithms, I simulate their behavior. I add a configurable jitter to the communication (Section 2.1.2). Specifically, I apply additional delay to a percentage of the packets, that is the percentage of the packets that are retransmitted and experience longer delay. Since TC does not offer a straightforward way to achieve this, I use iptables [64] to randomly mark a percentage of the packets. I then use a TC filter to apply the additional delay solely to these marked packets.

To verify this method, I configure a node's interface with a fixed delay of 100 ms, various jitter values, and different packet percentages. Using the ping utility, I measure the delay between this configured node and an unmodified node, sending 50000 ECHO_REQUEST packets. I count the affected packets and use this data to calculate the percentage of impacted packets.

Tables 3.5 and 3.6 display the calculated percentages of affected packets and error rates

| Configured Jitter Percentage | Measured Jitter Percentage | Error Rate |
|---|---|---|
| 0% | 0% | 0% |
| 1% | 0.98% | 2% |
| 20% | 19.92% | 0.4% |
| 50% | 49.79% | 0.42% |
| 80% | 79.98% | 0.02% |
| 100% | 99.97% | 0.03% |

Table 3.5: Verifying percentage of 5 ms jittered packets, standard deviation is less than 4%.

| Configured Jitter Percentage | Measured Jitter Percentage | Error Rate |
|---|---|---|
| 0% | 0% | 0% |
| 1% | 1% | 0% |
| 20% | 19.95% | 0.25% |
| 50% | 50.05% | 0.1% |
| 80% | 79.94% | 0.08% |
| 100% | 99.97% | 0.03% |

Table 3.6: Verifying percentage of 10 ms jittered packets, standard deviation is less than 5%.

for 5 ms and 10 ms jitter, respectively. Each measured jitter percentage is an average of five trials. The results show that the measured jitter percentages closely match the configured values. For 5 ms jitter, the deviation between the configured and measured values range between 0.04% and 2%, while for 10 ms jitter, it varies from 0.03% to 0.25%. These discrepancies are acceptable as marking packets in iptables follows a random model. These consistent results increase my confidence in the accuracy of the configuration, allowing me to trust that the setup is functioning as intended.

It is important to note that in a 5G environment, this additional delay can affect both uplink and downlink traffic. Consequently, when simulating a 5G link, I apply this jitter configuration to both the sending and receiving nodes.

## 3.2 TCP and Buffer Sizes

In my early experiments, throughput was unexpectedly low even when there were no packet drops. For example, Figure 3.1 shows the link utilization ratio with 50 ms RTT and varying the link bandwidth. Link utilization is the ratio of the measured throughput to the link bandwidth. With a bandwidth of 1 Gbps and 10 Gbps, the percentage of link utilization is very low even though there is no packet drop on the links.

My hypothesis is that this lower throughput is due to limited buffers at the sender and receiver. To verify this hypothesis, I run an experiment on a 1 Gbps and 50 ms RTT link and vary the TCP buffer sizes on the client and server nodes. Figure 3.2 shows throughput for different buffer sizes at a 1 Gbps and 50 ms RTT link. The results indicate that small buffer sizes have a direct impact on throughput. With the default buffer size of 4 MB, the experiment only achieves 327 Mbps (32.7% of the link bandwidth). TCP can saturate the link with buffers of 12 MB or larger. For the rest of this evaluation, I take a conservative approach. Unless otherwise stated, I configure TCP with 2 GB buffers to eliminate any impact that buffer limitations might have on throughput.

## 3.3 Choice of Congestion Control Protocols

There are numerous congestion control protocols available. Each protocol differs in the way it infers and reacts to congestion episodes. In fact, there are 17 congestion protocols natively supported by the Linux operating system. These techniques normally fall into four main categories as explained in Chapter 2: delay-based, loss-based, hybrid, and ECN-based. For my evaluation, I select the protocol that achieves the highest throughout for

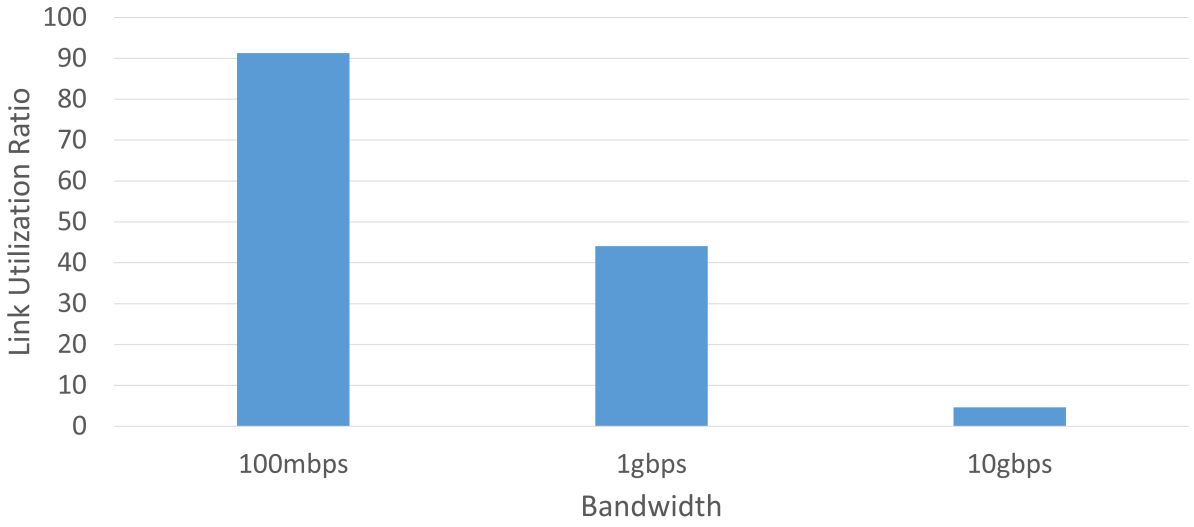



Figure 3.1: Average link utilization ratio with limited buffers for different link bandwidths at 50 ms RTT.

each category as a representative protocol for that category. To this end, I conduct an experiment to compare all the available congestion protocols in Linux, grouped by their categories. This experiment runs on 1 Gbps links, with 0.5% loss rate, and under three RTT delays: 1 ms, 50 ms, and 150 ms.

Figure 3.3, Figure 3.4, and Figure 3.5 show the throughput for different congestion protocols at 1 ms, 50 ms, and 150 ms RTT, respectively. The results in Figure 3.3 show that for delay-based protocols, BBR achieves the highest throughput. The second-best performing protocol is TCP LP. For loss-based protocols, Westwood achieves the highest throughput. Illinois achieves the highest throughput among the hybrid congestion protocol.

Comparing the results of 1 ms (Figure 3.3), 50 ms (Figure 3.4), and 150 ms (Figure 3.5) shows that for delay-based protocols, BBR remains the best-performing delay-based protocol across all RTT values. In the 50 ms scenario (Figure 3.4), there is no significant statistical difference between TCP LP and TCP NV, while TCP NV performs second-best to BBR in the 150 ms case. Westwood is still the best loss-based protocol in the 50 ms case, and there is no significant statistical difference between it and BIC. In the 150 ms case, TCP Hybla beats the other loss-based protocols. TCP Illinois remains the best hybrid congestion protocol in all of the scenarios.

Based on the results of this experiment, I select the best-performing protocols in each

category for most of the scenarios. For delay-based protocols, I choose TCP BBR, as it consistently achieves the highest throughput. However, BBR is not widely adopted due to issues discussed in Section 3.4. Therefore, for the delay-based protocols category, I also include the second-best performer, TCP LP. For loss-based protocols, I select TCP Westwood and TCP CUBIC, the latter being the default in Linux systems. For hybrid protocols, I pick TCP Illinois. For ECN-based protocols, I select DCTCP as it is the only candidate.

To summarize, here are the selected congestion control protocols:

- Delay-based: BBR and LP.
- Loss-based: Westwood and CUBIC.
- Hybrid: Illinois.
- ECN-based: DCTCP

## 3.4 BBRv1 vs BBRv3

BBR [13] is a congestion control algorithm designed by Google that deviates from using packet loss or buffer occupancy as indications of congestion. It utilizes Kleinrock's [40] formal model of congestion and its optimal operating point. It continuously measures the bottleneck bandwidth and the minimum RTT to approximate the congestion-free sending rate. With this, BBR intends to obtain low latency and high throughput, even in variable network conditions, without causing unreasonable queuing or bufferbloat.

In the results presented in the previous section, BBR achieves the highest throughput among the congestion protocols, yet it is not widely adopted. This is because BBR suffers from practical limitations, including elevated queuing delays and RTT unfairness. Moreover, it shows an immense amount of packet drop and unfairness to other flows with loss-based congestion control [34]. There are research efforts that try to enhance its fairness [61], reduce the excessive retransmissions [51], and modify the protocol to reduce the latency in fast-varying connections [15]. Cao et al. [12] present an empirical analysis and evaluation study of BBR. Their results indicate that BBR performs well in networks with shallow buffers, in spite of its high retransmissions. However, in networks with large buffers, conventional loss-based algorithms are more effective. Their analysis also reveals that BBR exhibits immense queue sizes. The scenarios where BBR is well-suited are often

the same ones where it is unfair to other competing flows. Atxutegi et al. [9] find that in mobile networks and under certain network conditions, longer BBR flows are unfair to shorter BBR flows.

Google then released BBRv2 [24], in which they claim to have fixed the issues of the original BBR algorithm (now referred to as BBRv1). BBRv2 incorporates packet loss in inferring congestion. However, BBRv2 is not available in a supported kernel release. In July 2023, BBRv3 [25] [23] was released, with plans to submit it for inclusion in the mainline Linux TCP stack.

As BBR's development progresses, there has been a debate about whether it is ready for broad adoption. Neal Cardwell, one of the key engineers at Google behind BBR, expressed his reservations, stating, "BBRv1 is definitely not ready to be the default on any Linux distribution. Whether BBRv3 is ready to be a distribution default is arguable. Personally, I would prefer to see BBRv3 improve with respect to Reno/CUBIC coexistence and keeping loss rates low before arguing that it is ready to be the default".

I conducted a follow-up experiment using BBRv3 to evaluate its performance in comparison to BBRv1. The results show that BBRv3 still maintains comparable performance to BBRv1. Consequently, my conclusions here on BBRv1 should hold for BBRv3.

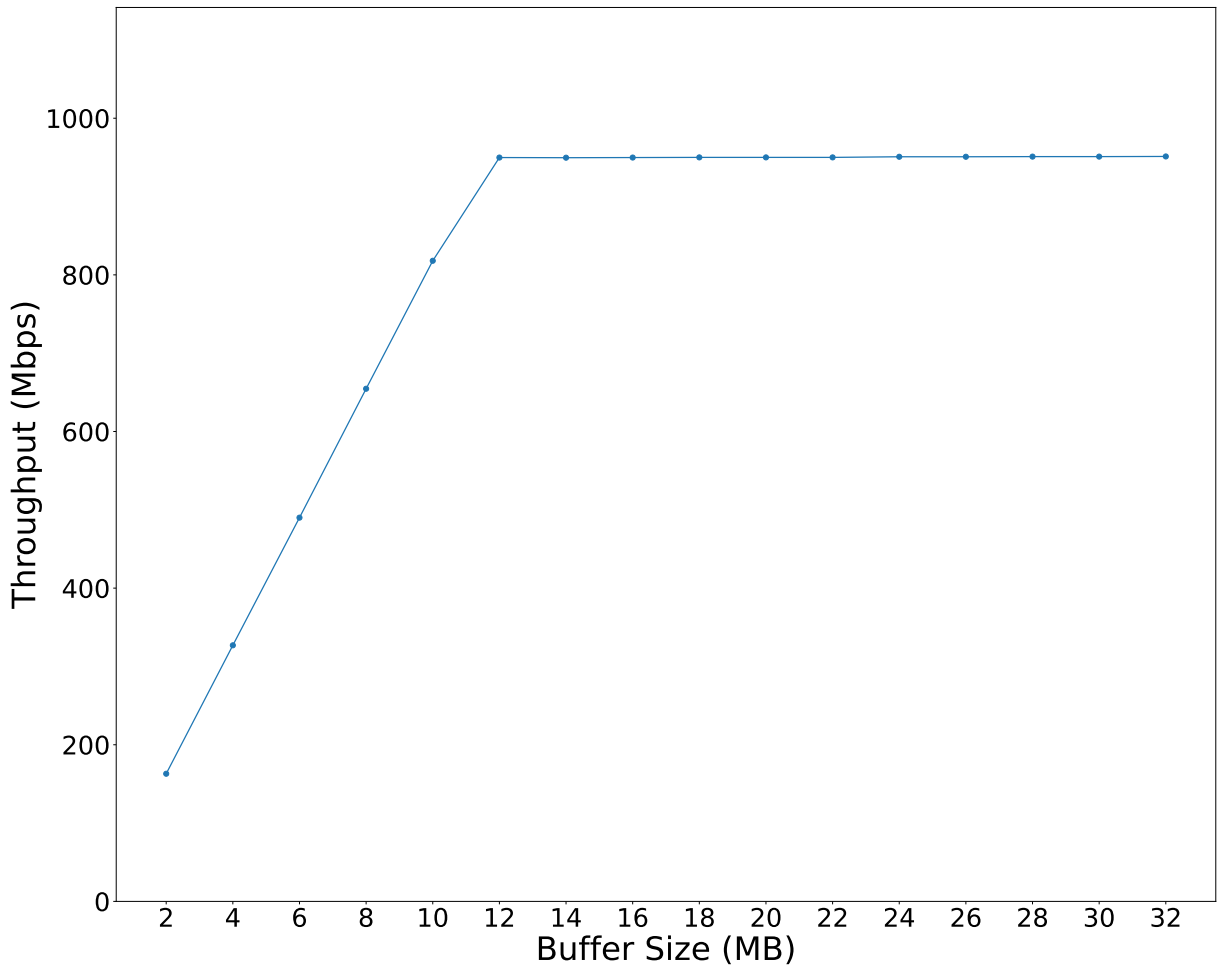



Figure 3.2: Average throughput for different buffer sizes for a 50 ms RTT and 1 Gbps link.

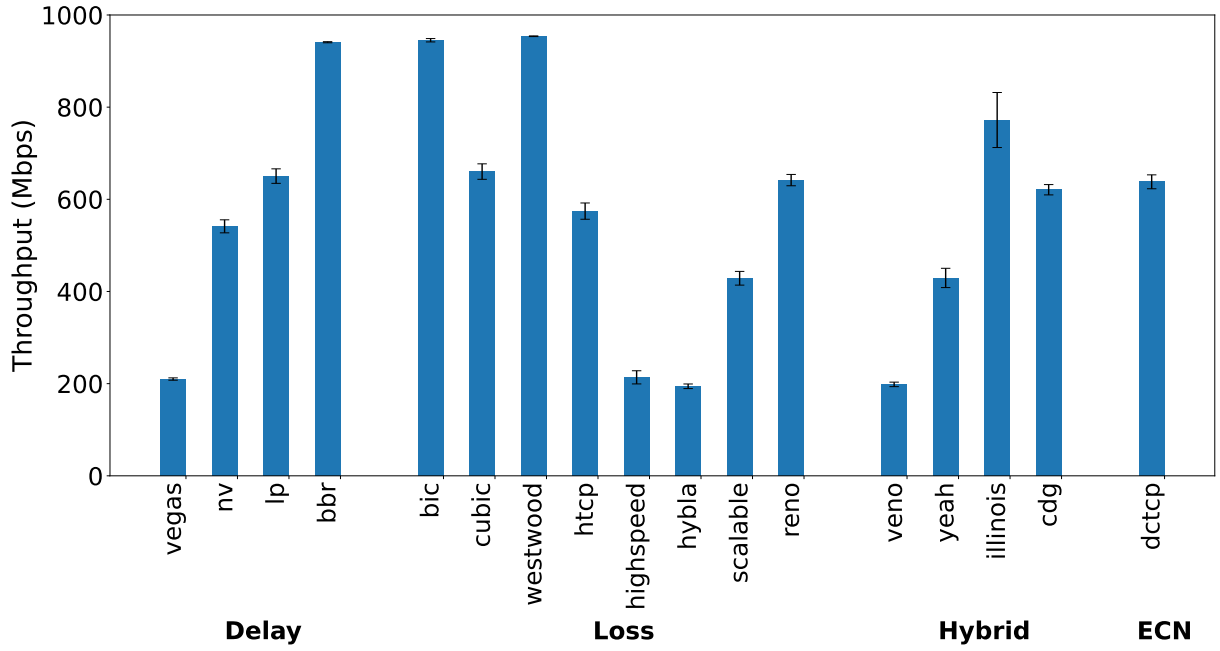



Figure 3.3: Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 1 ms RTT link, the error bars show the standard deviation.

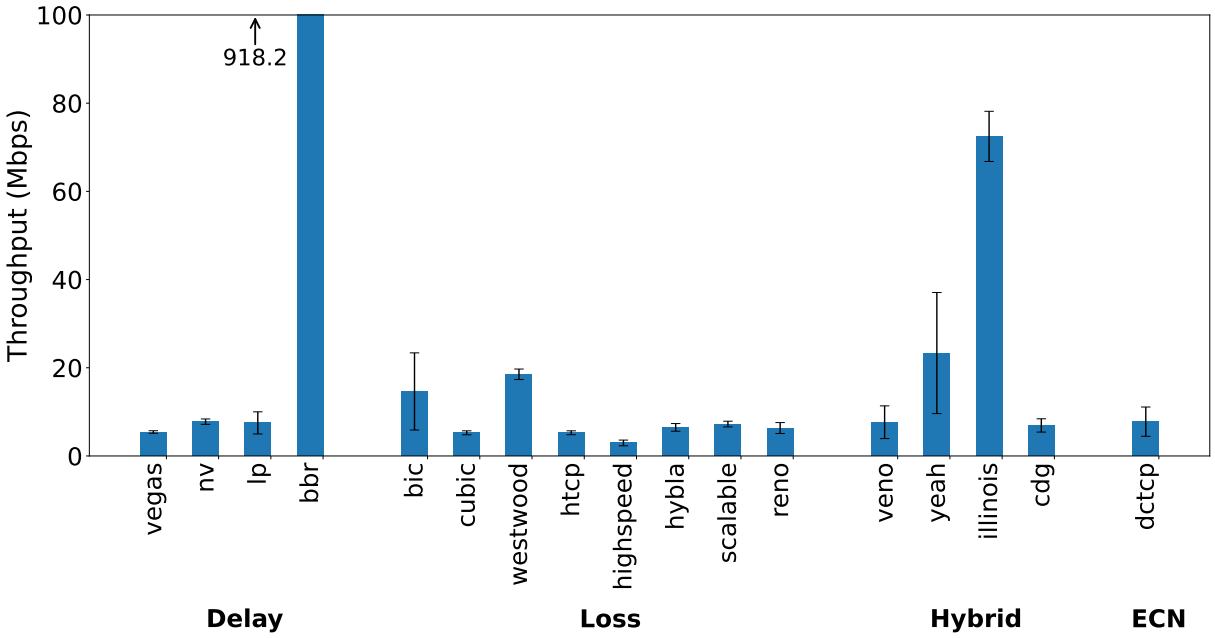



Figure 3.4: Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 50 ms RTT link, the error bars show the standard deviation.

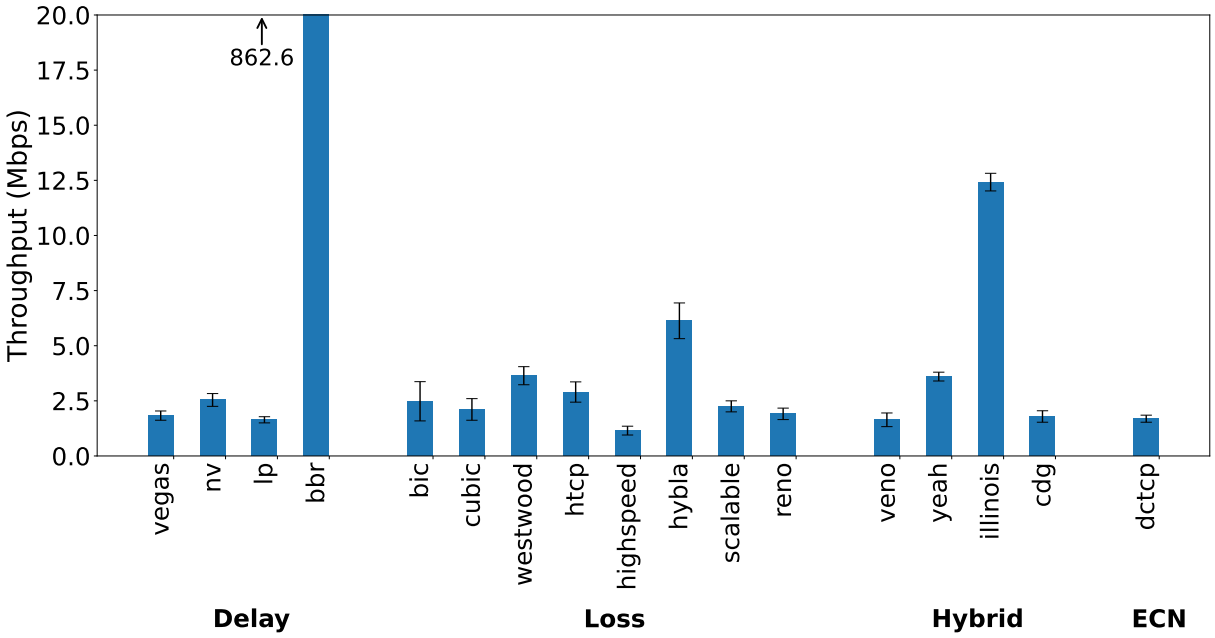



Figure 3.5: Average throughput of different congestion control protocols over a 1 Gbps, 0.5% loss and 150 ms RTT link, the error bars show the standard deviation.

# Chapter 4

# Evaluation

In this chapter, I present the experiments to examine the impact and viability of edge computing for improving end-to-end performance in WANs and 5G networks. I evaluate three main cases:

- Edge as an Accelerator for 5G Communication.
- Deploying Services on MEC.
- Edge as an Accelerator for WAN Communication.

The rest of this chapter studies each use case under different network conditions and protocols.

## 4.1 Use Case I: Edge as an Accelerator for 5G Communication

Edge clusters can accelerate end-to-end TCP throughput by using TCP splitting. Figure 4.1 shows this use case. The data center hosts the service, and the MEC that is close to the user acts as a TCP proxy, while the client runs the application. The latency between the end user device and the MEC is in the range of 1 ms to 5 ms, while the latency between the MEC and the data center can be 20 - 200 ms. With TCP splitting, there are two TCP connections: a short-distance connection in the radio network and a long-distance connection between the MEC and the data center. The short radio connection

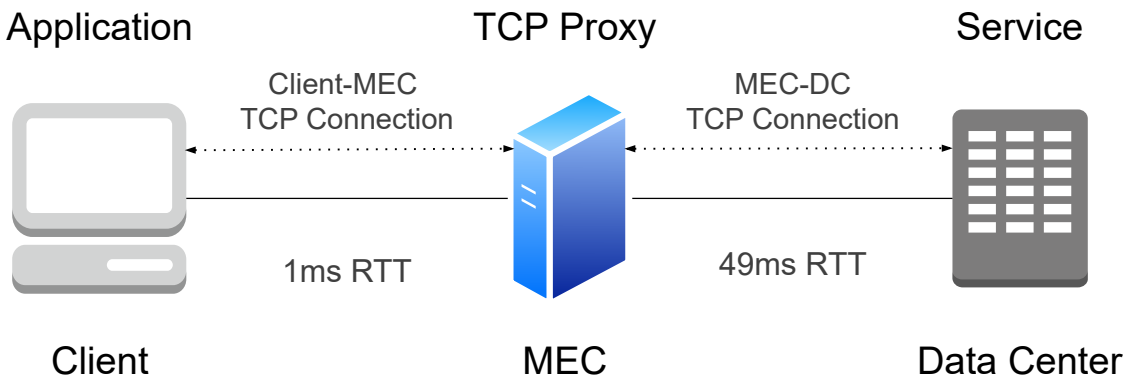



Figure 4.1: Edge as an accelerator for 5G communication.

has higher drop rate but due to its short RTT TCP performance can recover quickly. The long-distance connection has higher RTT but is more stable and experiences fewer packet drops. This approach can reduce latency and significantly improve end-to-end throughput.

I study this use case using both the classical and 5G communication stacks. In the classical communication stack, packet drops result in packet loss within TCP streams, whereas in the 5G communication stack, packet drops manifest as jitter (Chapter 2).

To explore the benefits of TCP splitting in this context, I address the following key questions:

- What is the impact of packet drop on end-to-end throughput over the classical communication stack while varying the MEC placement?
- What is the impact of packet drop on end-to-end throughput over the 5G communication stack?
- What is the impact of TCP buffer sizes on end-to-end throughput? Are the default buffer values enough?
- How do different congestion control protocols perform over both the classical and 5G communication stacks?
- Under what traffic conditions does TCP splitting enhance the performance? For instance, at what flow size does TCP splitting bring performance benefits?

Unless otherwise specified, I conduct the experiments on a 1 Gbps link with the maximum TCP buffer configuration, using TCP CUBIC, the default congestion control protocol in Linux systems.

### 4.1.1 Impact of Packet Drop Over The Classical Communication Stack

This experiment aims to study the impact of packet drop when the MEC uses TCP splitting to accelerate the communication to the nearest data center. The experiment runs on a 50 ms link with a bandwidth of 1 Gbps. I deploy a TCP proxy at 1 ms and 5 ms from the client, and the packet drop rate varies across 0%, 0.05%, 0.1%, 0.5%, and 1%.

Figure 4.2 shows the throughput for different packet drop rates for a proxy placed at 1 ms RTT. The figure shows that throughput drops as the drop rate increases. Without a proxy, throughput drops by 85.6% (from 949 Mbps to 136.21 Mbps) with 0.05% drop rate and drops by 99.6% (from 949 Mbps to 4.04 Mbps) with 1% drop rate. It is evident that a TCP proxy significantly improves the end-to-end throughput. The performance difference increases as the drop rate increases. A proxy deployed at 1 ms RTT from a user improves the end-to-end throughput by 7×, 17.4×, 77.6×, and 115.5× with drop rates of 0.05%, 0.1%, 0.5% and 1%, respectively.

Figure 4.3 shows the throughput for different packet drop rates for a proxy placed at 5 ms RTT. The results show a similar trend to the 1 ms setup; throughput drops as packet drop increases. A TCP proxy deployed at 5 ms RTT improves the end-to-end throughput by 17.4×, 34.8%, 20.8× and 14.9× with drop rates of 0.05%, 0.1%, 0.5% and 1%, respectively.

A comparison of the results for the 1 ms setup in Figure 4.2 and the 5 ms setup in Figure 4.3 shows that throughput is higher when placing the proxy closer to the client (at 1 ms RTT). With a drop rate of 0.05%, positioning the proxy at 1 ms achieves 27% better throughput (948.2 Mbps vs 746.3 Mbps) than at 5 ms. This improvement is due to the shorter feedback loop of the lossy connection, allowing TCP to recover faster when the MEC is closer to the client.

The experiments show that even a low drop rate in the RAN has a significant negative impact on TCP communication, and using a TCP proxy has the potential to significantly improve end-to-end performance. As the links become more unreliable, the benefit of deploying a TCP proxy increases significantly.

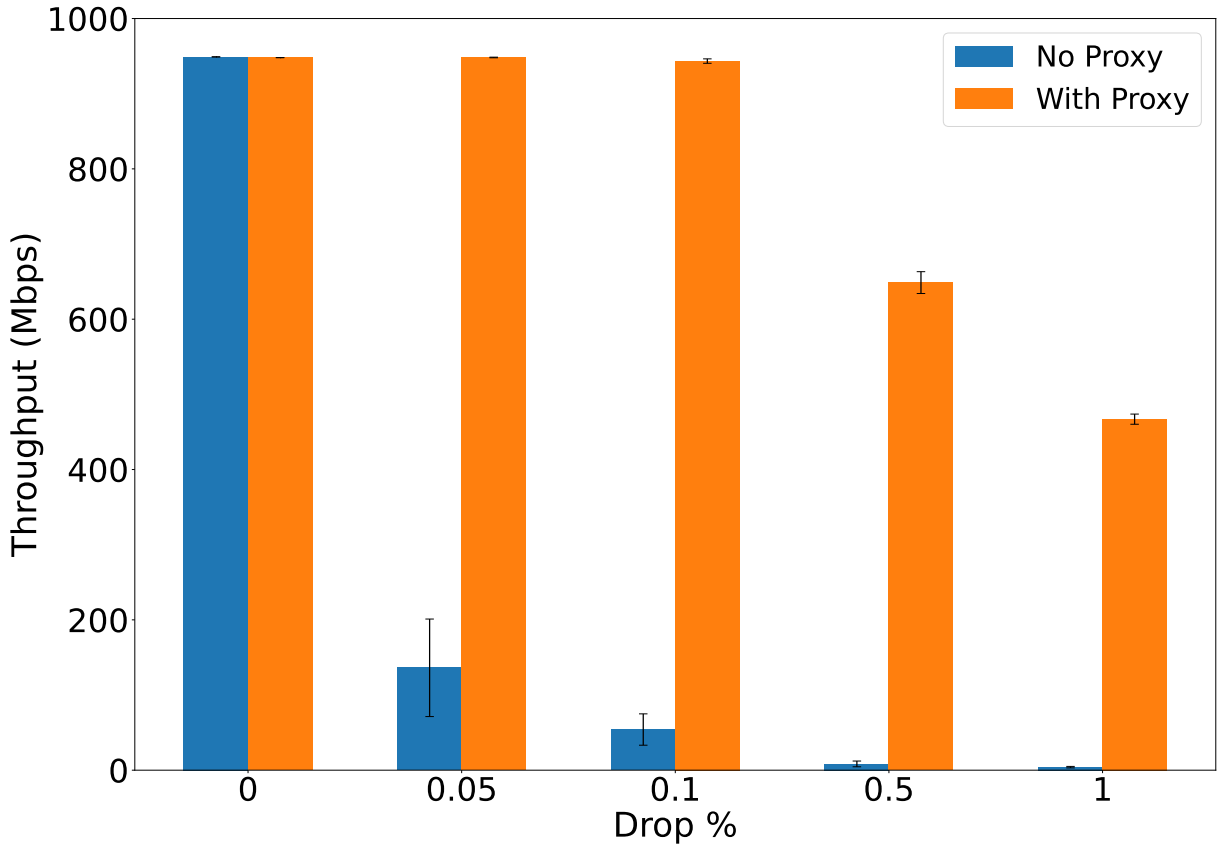



Figure 4.2: Average throughput for different drop rates at 50 ms RTT, 1 Gbps, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation.

### 4.1.2 Impact of Packet Drop Over The 5G Communication Stack

I study the impact of packet drop on this deployment over the 5G communication stack. I simulate the behavior of this communication stack by randomly introducing jitter to a percentage of the packets (Section 2.1.2). In this case, the TCP layer will experience random latency spikes.

I run an experiment on a 1 Gbps link with 50 ms RTT. I set jitter to 10 ms, based on the assumption that there is a significant load and that if the client is not scheduled to transmit immediately, it will be scheduled to transmit after 10 ms, in the next 5G frame (Section 2.1.2). The percentage of jittered packets determines how often the client will need to wait for the additional 10 ms, which I vary between 0%, 0.05%, 0.1%, 0.5%, and

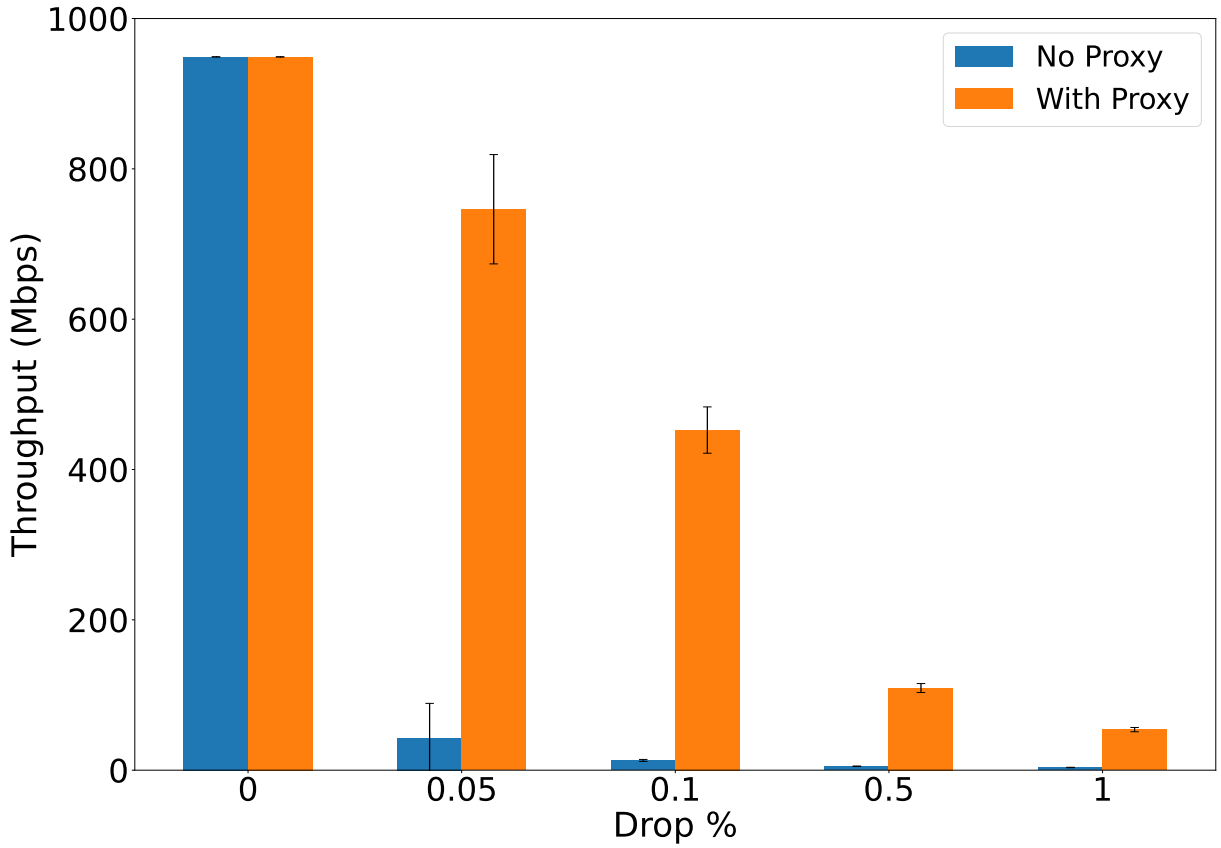



Figure 4.3: Average throughput for different drop rates at 50 ms RTT, 1 Gbps, and MEC at 5 ms over the classical communication stack. Error bars show the standard deviation.

1%. I deploy a TCP proxy at 1 ms and 5 ms from the client.

Figure 4.4 shows the average throughput for different percentages of jittered packets for a proxy at 1 ms RTT. The results indicate that TCP splitting is ineffective over the 5G communication stack between 0 and 0.1% drop rates. The throughput without a proxy remains robust because TCP does not encounter any losses, and the 10 ms latency spike is tolerable relative to the 50 ms RTT. A 20% increase in RTT time does not trigger the TCP congestion control mechanism. However, introducing TCP splitting in this context leads to performance degradation at 0.5% and 1% jittered packets. For a proxy at 1 ms, splitting the TCP connection results in 33.7% and 58.1% decreases in throughput at 0.5% and 1% jittered packets, respectively. This is because the additional 10 ms delay is substantial in the TCP proxy setup. The 10 ms delay represents 2× to 10× of the RTT on the short link

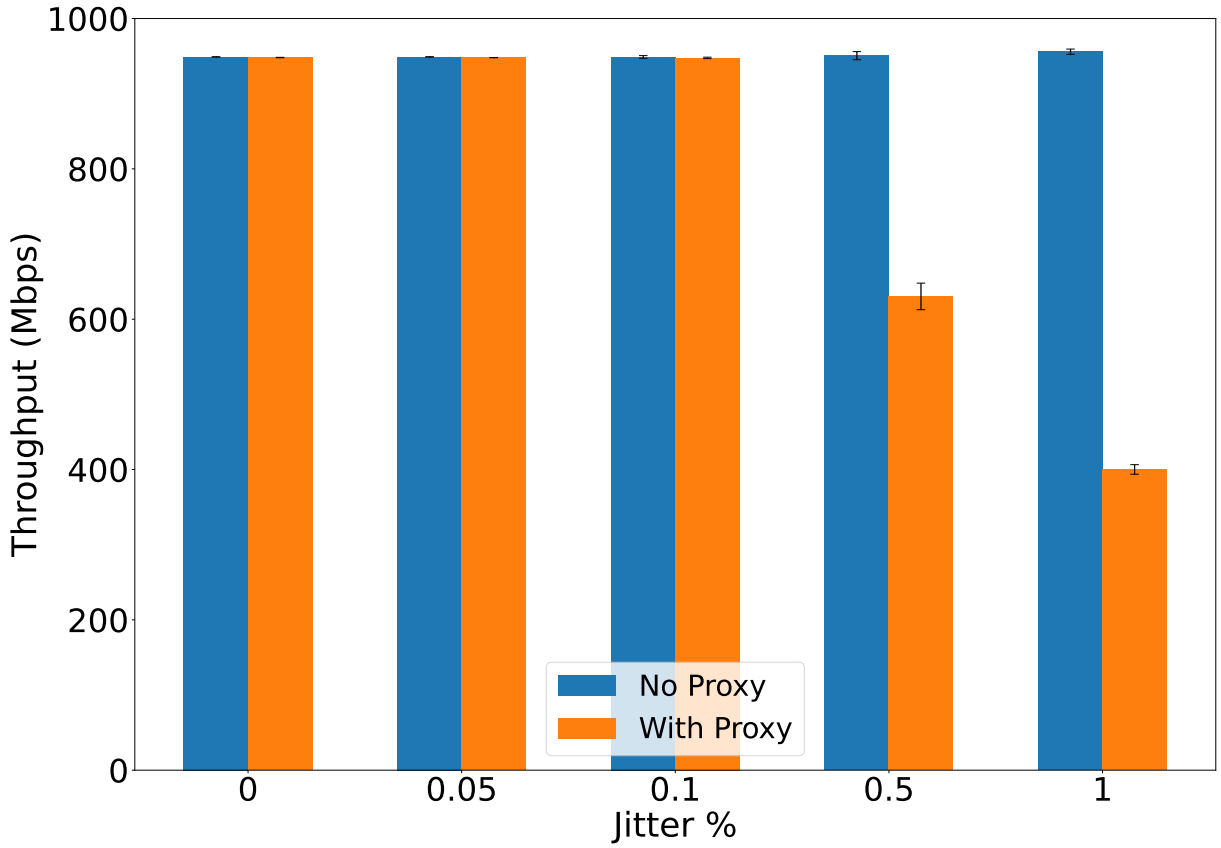



Figure 4.4: Average throughput for different percentages of jittered packets at 50 ms RTT, 1 Gbps, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation.

which has an RTT of 1-5 ms. This causes a significant increase in packet retransmissions at the TCP layer compared to the no-proxy scenario.

Figure 4.5 shows the average throughput for a proxy at 5 ms RTT. TCP splitting remains ineffective over the 5G communication stack. It decreases throughput by 85.7% and 91.8% at 0.5% and 1% jittered packets, respectively. The decrease in throughput with the TCP proxy in the 5 ms setup is higher than in the 1 ms setup due to the longer recovery time required for TCP.

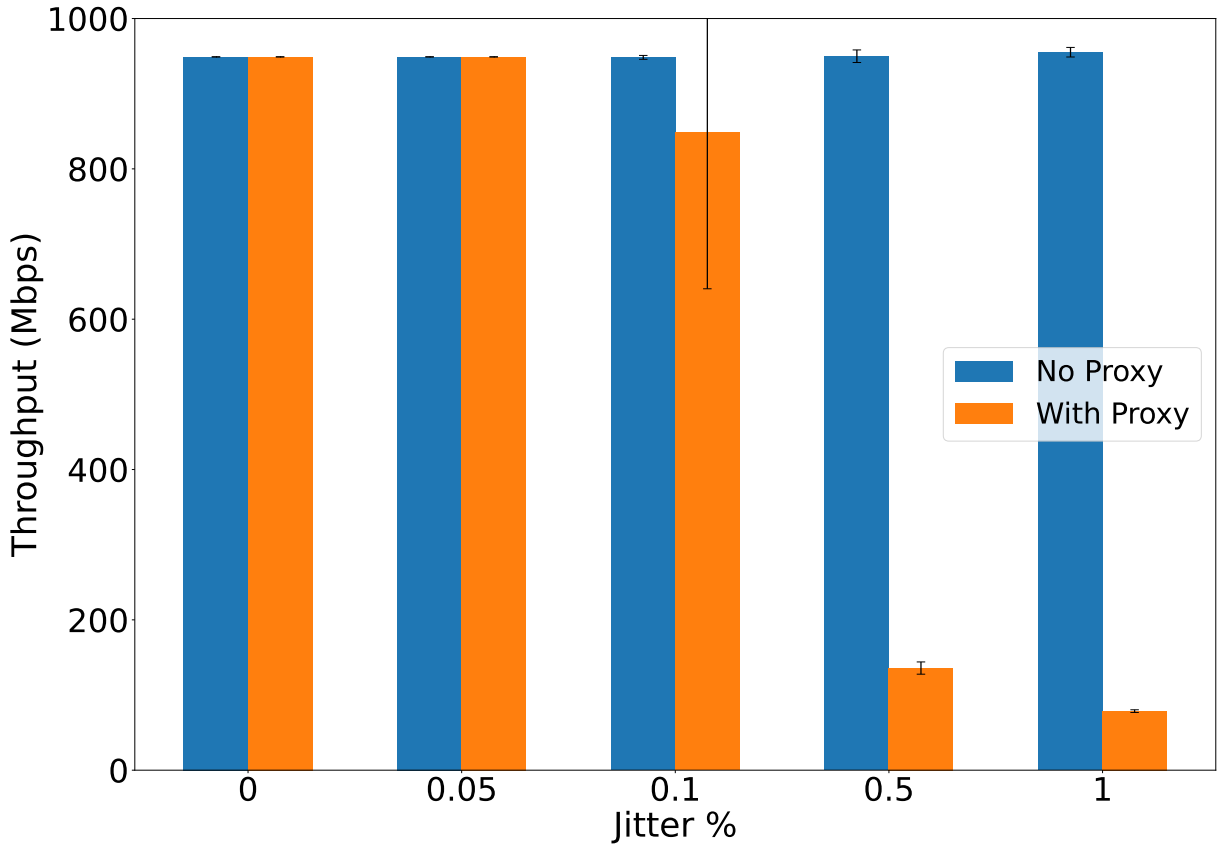



Figure 4.5: Average throughput for different percentages of jittered packets at 50 ms RTT, 1 Gbps, and MEC at 5 ms over the 5G communication stack. Error bars show the standard deviation.

### 4.1.3 Impact of TCP Buffer sizes over the Classical and 5G Communication Stacks

TCP buffer sizes are known to have a large impact on TCP throughput, especially in setups with large RTT [18]. In this section, I investigate the impact of TCP buffer sizes on the end-to-end throughput over both the classical and 5G communication stacks. The experiments fall into three categories: no packet drop, the classical communication stack, and the 5G communication stack. I configure the link with 50 ms RTT and 1 Gbps bandwidth. For the classical configuration stack, I add a packet loss rate of 0.5%, while for the 5G stack, I configure a 10 ms jitter affecting 0.5% of the packets. I deploy the MEC running the TCP

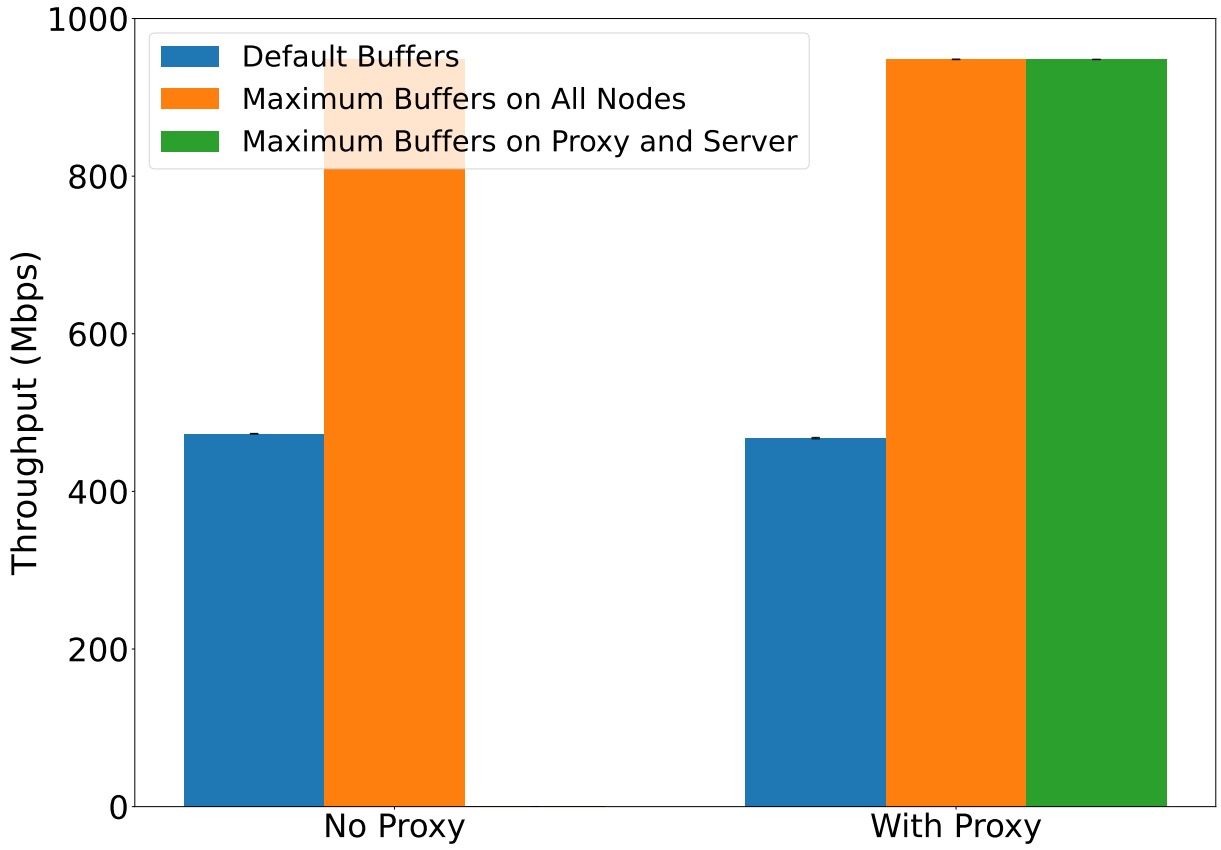



Figure 4.6: Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, and MEC at 1 ms. Error bars show the standard deviation.

proxy at 1 ms RTT from the client.

I experiment with three alternatives:

- The client, MEC, and data center have default TCP buffer sizes.
- The client uses the default buffer size, while MEC and data center are configured with maximum buffer sizes. This setup is practical when the service does not control client devices.
- All nodes are configured with maximum buffer sizes.

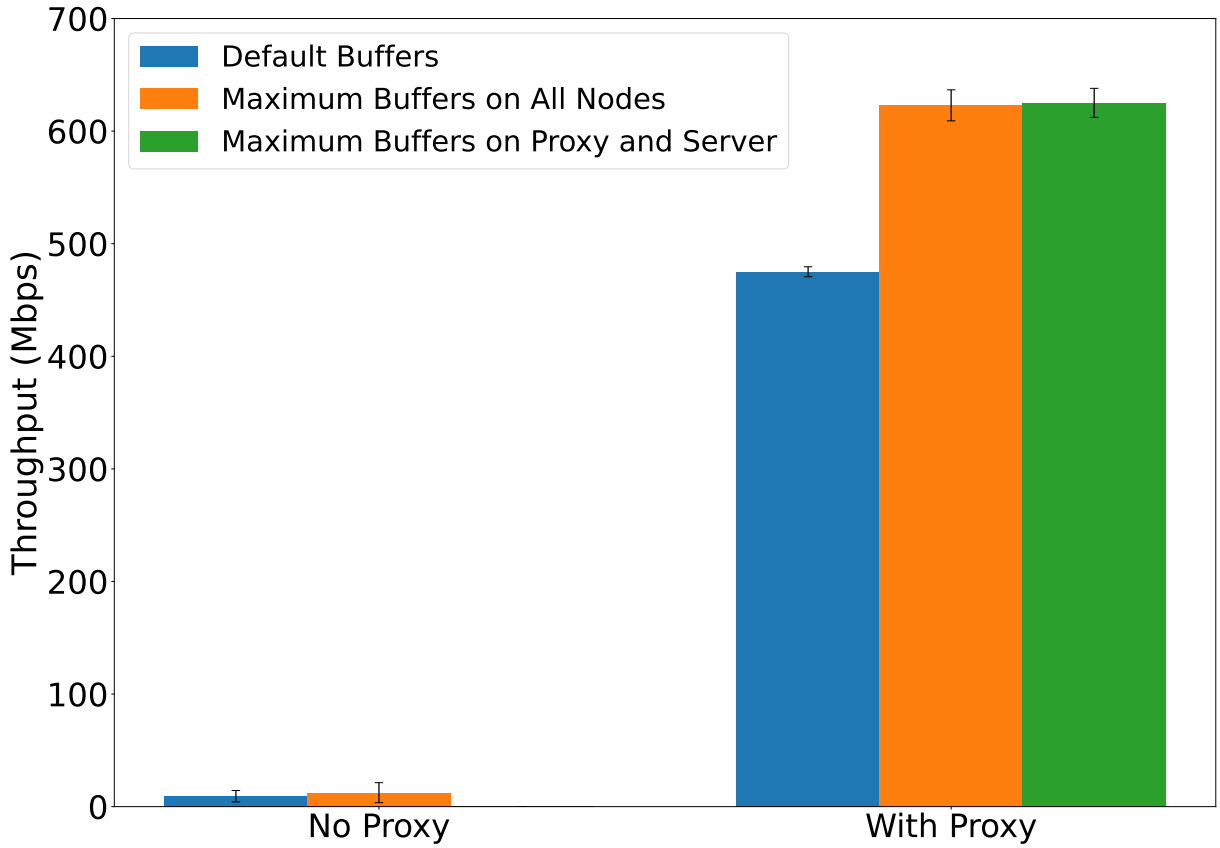



Figure 4.7: Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, 0.5% drop, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation.

Figure 4.6 shows the average throughput for different buffer configurations without packet drop. When there is no proxy, the default TCP buffers in Linux result in under-utilizing the link capabilities, with throughput reaching only 473.2 Mbps, which is 47.32% of the link capacity. The same case with maximum buffers results in increasing the throughput, reaching 949 Mbps. With TCP splitting and default buffers, the throughput is 467.6 Mbps, which is comparable to the no proxy and default buffers case. This is because without losses, the 49 ms RTT proxy-to-server segment is still similar to a single 50 ms RTT link, and there will be no benefit from the 1 ms client-to-proxy segment. However, increasing the buffer sizes still causes the throughput to improve to 948.2 Mbps.

Figure 4.7 shows the average throughput for different buffer configurations over the

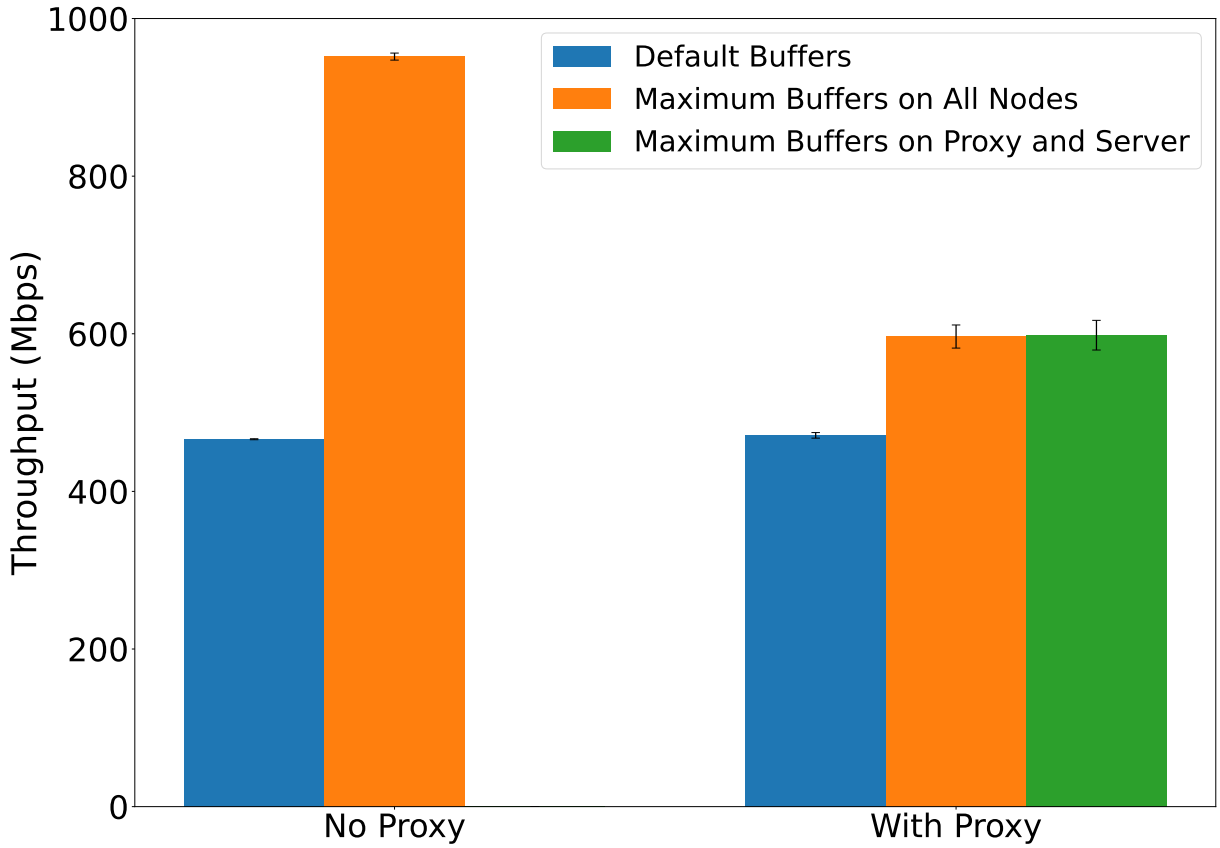



Figure 4.8: Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, 10 ms jitter affecting 0.5% of packets, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation.

classical communication stack with 0.5% packet drop rate. The results show that the network is still under-utilized with the default buffer configuration. Even with TCP splitting, the throughput is only 475.1 Mbps (47.51% of the link capacity). Although this is a significant improvement compared to the no proxy and default buffers case, the performance is still suffering from small buffers. Maximizing the buffers with TCP splitting increases the throughput by 31.1% compared to TCP splitting with default buffers. TCP splitting with maximum buffer sizes achieves 622.9 Mbps (62.29% of the link capacity).

It is important to note that in both the no drop and classical communication stack cases, increasing the buffers on the proxy and server nodes only, without modifying the client node, significantly improves throughput, and comparable to that of when modifying

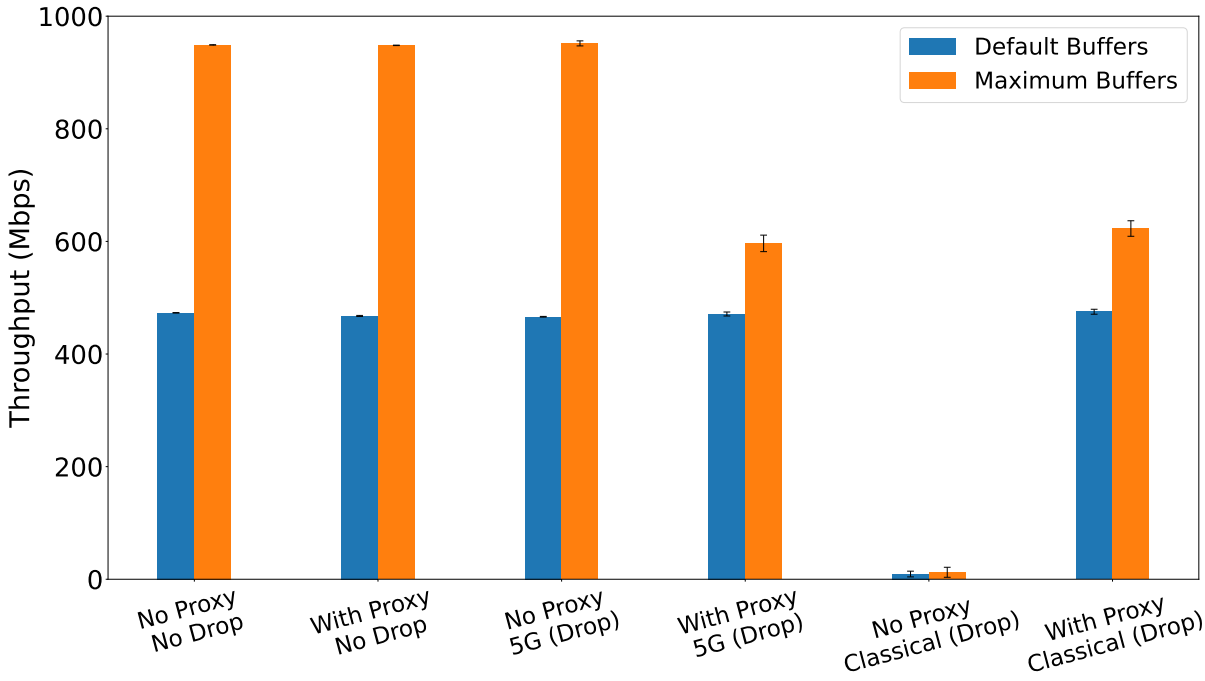



Figure 4.9: Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps over different communication stacks. Error bars show the standard deviation.

all of the nodes. This is because the proxy-server connection has as RTT of 49 ms, while the original connection has a 50 ms RTT, so the longer connection benefits greatly from the larger buffer sizes. Although the client connection is left with default buffers, its shorter RTT of 1 ms enables quicker recovery from packet drop.

Figure 4.8 shows the average throughput for different buffer configurations over the 5G communication stack. The network is still under-utilized with the default buffer configuration. Without TCP splitting, using the maximum TCP buffers increases throughput by 2×, reaching 95.17% of the link capacity. With a TCP proxy, larger buffers achieve around 26.6% higher throughput compared to the default buffer size configuration. Nevertheless, TCP splitting remains inefficient over the 5G communication stack. With maximum buffer sizes set on the data center, TCP splitting causes a 37% reduction in throughput compared to the no-proxy deployment with maximum buffers.

Figure 4.9 combines the throughput results for the three categories: no packet drop, the 5G communication stack, and the classical communication stack. I include two buffer configurations: default and maximum buffers on all nodes, as the results for maximum buffers

on the proxy and server are very similar to those with maximum buffers on all nodes. The figure shows that the no proxy case over the 5G communication stack is similar to the no drop cases. This is expected as the 5G communication stack hides packet drop from the TCP layer. Using maximum buffers in this case increases throughput by 2×. Throughput with proxy over the classical communication stack is comparable to throughput with TCP proxy over the 5G communication stack. With TCP splitting, using maximum buffers increases throughput by 31% over the classical communication stack, and 27% over the 5G communication stack. However, over the 5G communication stack and with maximum buffers, throughput with a proxy is still significantly less than that of without a proxy. Throughput is lowest in the no-proxy case over the classical communication stack. Increasing the buffer sizes does not significantly improve performance, highlighting the substantial impact of packet loss on TCP.

### 4.1.4 Behavior of Congestion Control Protocols Over the Classical and 5G Communication Stacks

To study the behavior of different congestion control protocols in the context of TCP proxy deployment on MEC, I conduct two experiments: one over the classical communication stack, and the other over the 5G communication stack. Both experiments run on a link with a 50 ms RTT and 1 Gbps bandwidth, and with a TCP proxy placed at 1 ms from the client. In the classical communication stack experiment, the packet loss rate is 0.5%, while in the 5G stack experiment, 10 ms jitter affects 0.5% of the packets. The congestion control protocols evaluated are BBR, LP, Westwood, CUBIC, Illinois, and DCTCP (Chapter 3).

Figure 4.10 shows the average throughput for different congestion control protocols over the classical communication stack. The results show that every congestion control protocol, except for BBR, benefits significantly from TCP splitting. The throughput increases are significant, with CUBIC, Westwood, LP, Illinois, and DCTCP experiencing gains of 127×, 51×, 86×, 13×, and 84×, respectively. With the use of the proxy, Westwood, Illinois, and BBR achieve the highest throughput, then CUBIC and DCTCP, followed by LP. BBR achieves 918 Mbps without TCP splitting, and using the proxy has minimal effect in this scenario. This suggests that TCP splitting is unnecessary and that using BBR alone is sufficient. However, as discussed in Section 3.4, BBR has several issues and is not yet ready for widespread adoption.

Figure 4.11 shows the average throughput for different congestion control protocols over the 5G communication stack. As expected, TCP splitting results in performance degradation over the 5G communication stack. However, with the proxy enabled, Westwood,

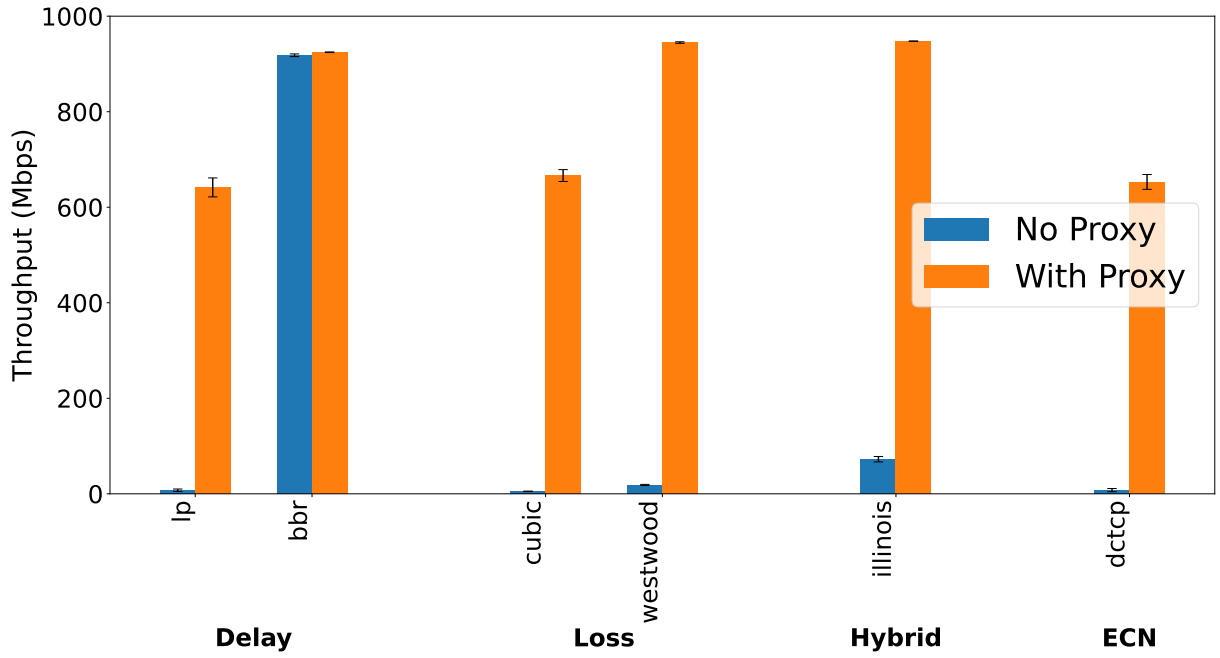



Figure 4.10: Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 0.5% drop, and MEC at 1 ms over the classical communication stack. Error bars show the standard deviation.

Illinois, and BBR still achieve results close to those in the no-proxy case. The results show that all protocols perform well and similarly without a proxy, which is expected since TCP perceives no losses over the 5G communication stack.

### 4.1.5 Impact of TCP Flow Size Over The Classical Communication Stack

TCP splitting significantly improves performance over the classical communication stack when transmitting large amounts of data. In this section, I investigate the flow sizes at which TCP splitting is beneficial.

I conduct an experiment to measure the time from the moment a TCP socket is opened until the socket is closed successfully, this includes the time to send the data over the network. I repeatedly measure this time for different data sizes. For each size, I run the experiment for at least 60 seconds or until I send the data at least 10 times, then I average

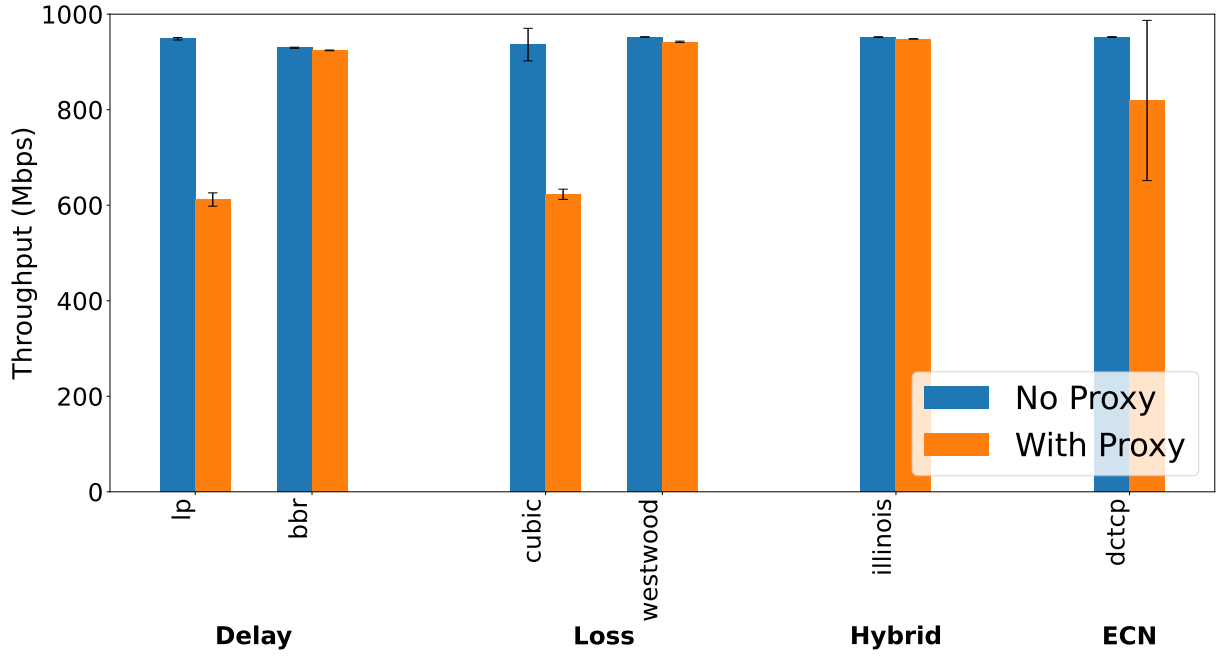



Figure 4.11: Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 10 ms jitter affecting 0.5% packets, and MEC at 1 ms over the 5G communication stack. Error bars show the standard deviation.

these measurements. I refer to this average as the response time. I measure the response time because smaller flow sizes can not fully utilize the network capacity.

I run the experiment on a 1 Gbps link with 50 ms RTT and 0.5% packet drop. Flow sizes vary from 1 KB to 1 GB, with placing the MEC at 1 ms and 5 ms form the client.

Figure 4.12 shows the average response time for different data sizes over the classical communication stack. For data sizes from 1 KB - 64 KB, the response times in the proxy cases are significantly less than than the original connection (without TCP splitting). This is expected since in the proxy cases, the proxy is so close to the client that transmission to the proxy is done quickly and the socket is closed quickly. After that, as the size increases, the response time increases. With the use of a TCP proxy, the response time is less than the with-drop-no-proxy case. Starting from 4 MB, proxy at 1 ms gets close to the no-proxy-no-drop case.

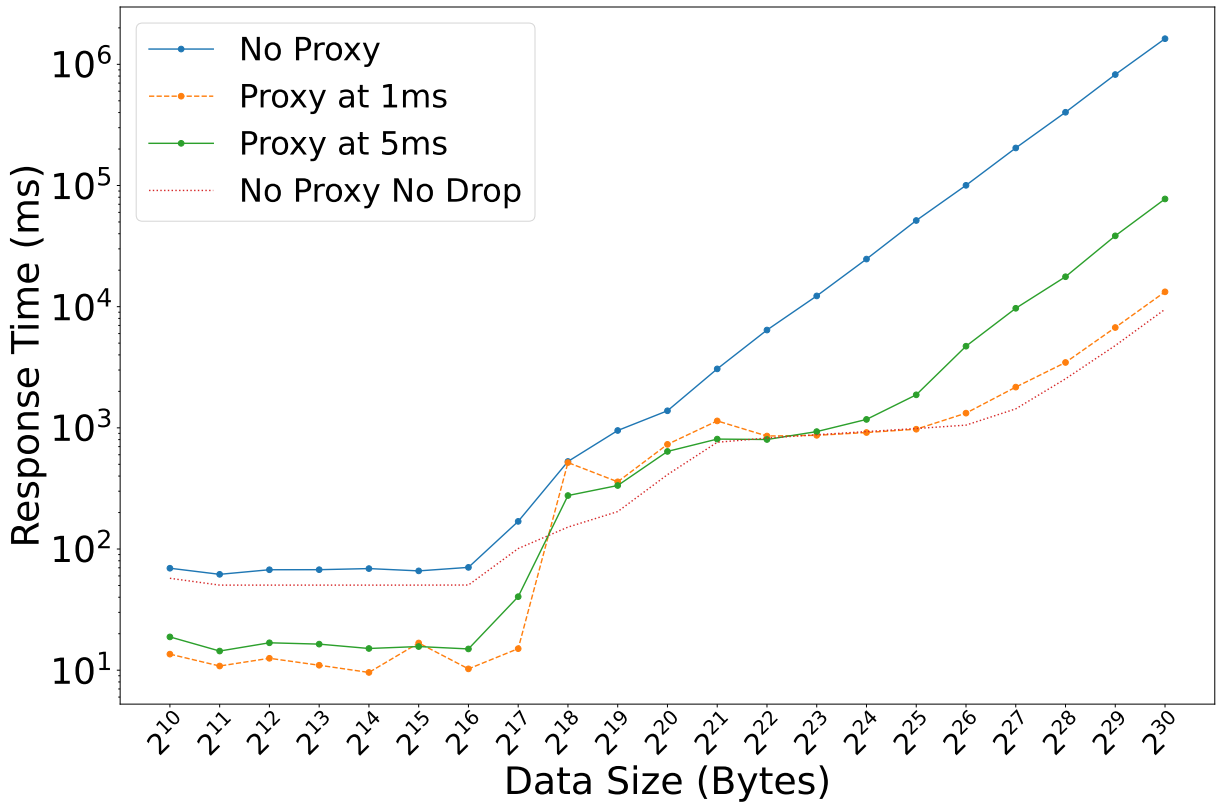



Figure 4.12: Average response time for different flow sizes at 50 ms RTT, 1 Gbps, and 0.5% drop over the classical communication stack.

### 4.1.6 Summary

TCP splitting significantly improves end-to-end throughput over the classical communication stack, especially as packet drop rates increase. Throughput is higher when the proxy is deployed very close to the client (1 ms RTT). However, over the 5G communication stack, TCP splitting is ineffective. The 5G communication stack causes latency spikes and hides packet drop from the TCP layer. Without a TCP proxy, this latency spike is around 20% of the original RTT, and will not reduce the congestion window. With a TCP proxy, this latency spike is 2× to 10× the original RTT, leading to reductions in the congestion window and consequently degrading TCP performance.

TCP Buffer sizes significantly impact performance. The default TCP buffers do not fully utilize the network capabilities. Over the classical communication stack, increasing

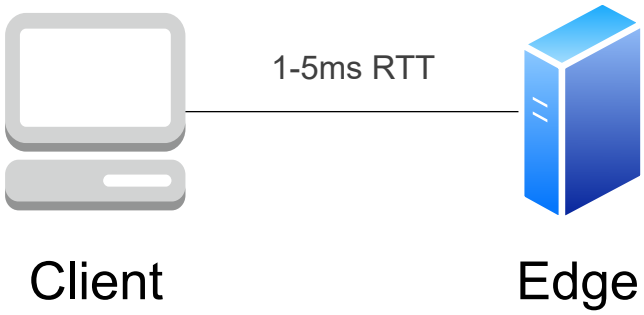



Figure 4.13: Deploying services on MEC.

buffers on the proxy-server connection is as good as increasing the buffers on all nodes. Although the client connection uses default buffer sizes, its shorter RTT of 1 ms enables quicker recovery from packet drop. This is beneficial because increasing buffer sizes on clients is often challenging due to limited resources and lack of privileged access to modify client configurations.

Congestion control protocols perform differently. Over the classical communication stack, all congestion protocols benefit greatly from splitting the TCP connection, except for BBR. The 5G communication stack causes all the congestion protocols to perform well without the need for TCP splitting. However, with TCP splitting over the 5G communication stack, throughput either remains the same or declines.

Over the classical communication stack, splitting the TCP connection brings performance benefits for flow sizes over 64 KB. With flow sizes of 64 KB or less, the response time is low because the client-to-proxy connection is very short.

## 4.2 Use Case II: Deploying Services on MEC

In this scenario, the edge hosts the complete service. Figure 4.13 shows this use case. The service is deployed on the 5G MEC that is close to the end user device. The latency between the end user device and the MEC can be in the range of 1 ms to 5 ms.

I study this use case using both the classical and 5G communication stacks. In the classical communication stack, packet drops result in packet loss within TCP streams, whereas in the 5G communication stack, packet drops manifest as jitter (Chapter 2).

I study the performance implications of running services on the MEC. Specifically, I explore the following questions:

- What is the impact of packet drop on end-to-end throughput over the classical communication stack?
- What is the impact of packet drop on end-to-end throughput over the 5G communication stack?
- How do different congestion control protocols perform over both the classical and 5G communication stacks?

Unless otherwise specified, I conduct the experiments on a 1 Gbps link with the maximum TCP buffer configuration, using TCP CUBIC, the default congestion control protocol in Linux systems.

### 4.2.1 Impact of Packet Drop Over The Classical Communication Stack

In this experiment, I study the impact of packet drop on throughput over client-to-MEC connections. The experiment runs on links with 1 ms and 5 ms RTTs, each with a bandwidth of 1 Gbps. The packet drop rate varies across 0%, 0.05%, 0.1%, 0.5%, and 1%.

Figure 4.14 shows the average throughput for different drop rates at 1 ms RTT. Throughput decreases as the packet drop rate increases beyond 0.1%. This decline is not linear; higher drop rates result in more pronounced throughput degradation. This behavior is expected as the relationship between throughput and packet loss is nonlinear.

Figure 4.15 shows the average throughput for different drop rates at 5 ms RTT. Throughput decreases as the packet drop rate increases. The overall throughput is lower compared to the 1 ms case, and the throughput decline is more pronounced. This suggests that higher RTTs can aggravate the impact of losses on throughput. Both figures emphasize the sensitivity of throughput to loss, particularly under different RTT conditions.

### 4.2.2 Impact of Packet Drop Over The 5G Communication Stack

This experiment studies how the 5G communication stack affects the performance in this deployment scenario. I simulate the behavior of this communication stack by randomly

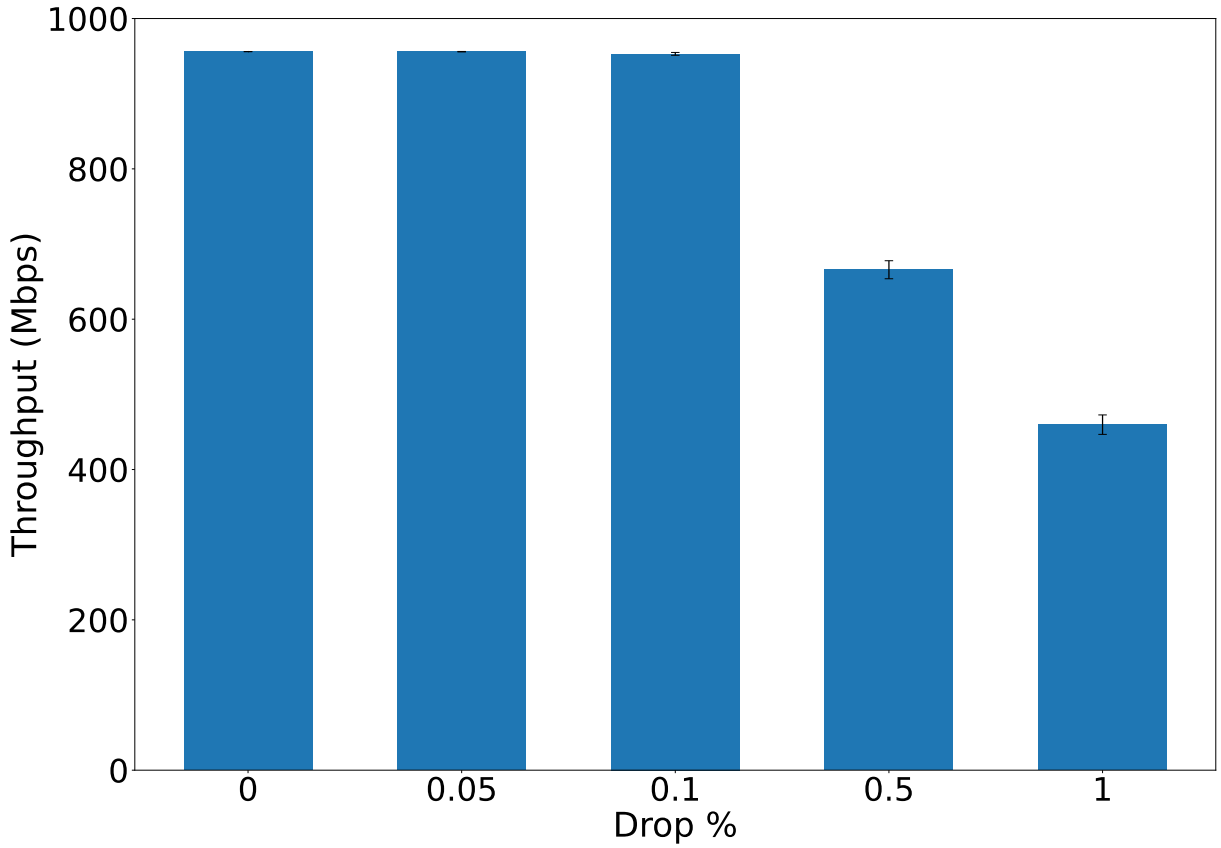



Figure 4.14: Average throughput for different drop rates at 1 ms RTT and 1 Gbps over the classical communication stack. Error bars show the standard deviation.

introducing jitter to a percentage of the packets (Section 2.1.2). The TCP layer will not experience losses, but there will be random latency spikes.

I conduct this experiment on 1 Gbps links with 1 ms and 5 ms RTTs. I set jitter to 10 ms, based on the assumption that there is a significant load and that if the client is not scheduled to transmit immediately, it will be scheduled to transmit after 10 ms, in the next 5G frame (Section 2.1.2). The percentage of jittered packets determines how often the client will need to wait for the additional 10 ms, which I vary between 0%, 0.05%, 0.1%, 0.5%, and 1%.

Figure 4.16 presents the average throughput for different percentages of jittered packets for 1 ms RTT. Throughput remains stable between 0% and 0.1% jittered packets. However, as the percentage of jittered packets increases beyond 0.1%, the throughput decreases

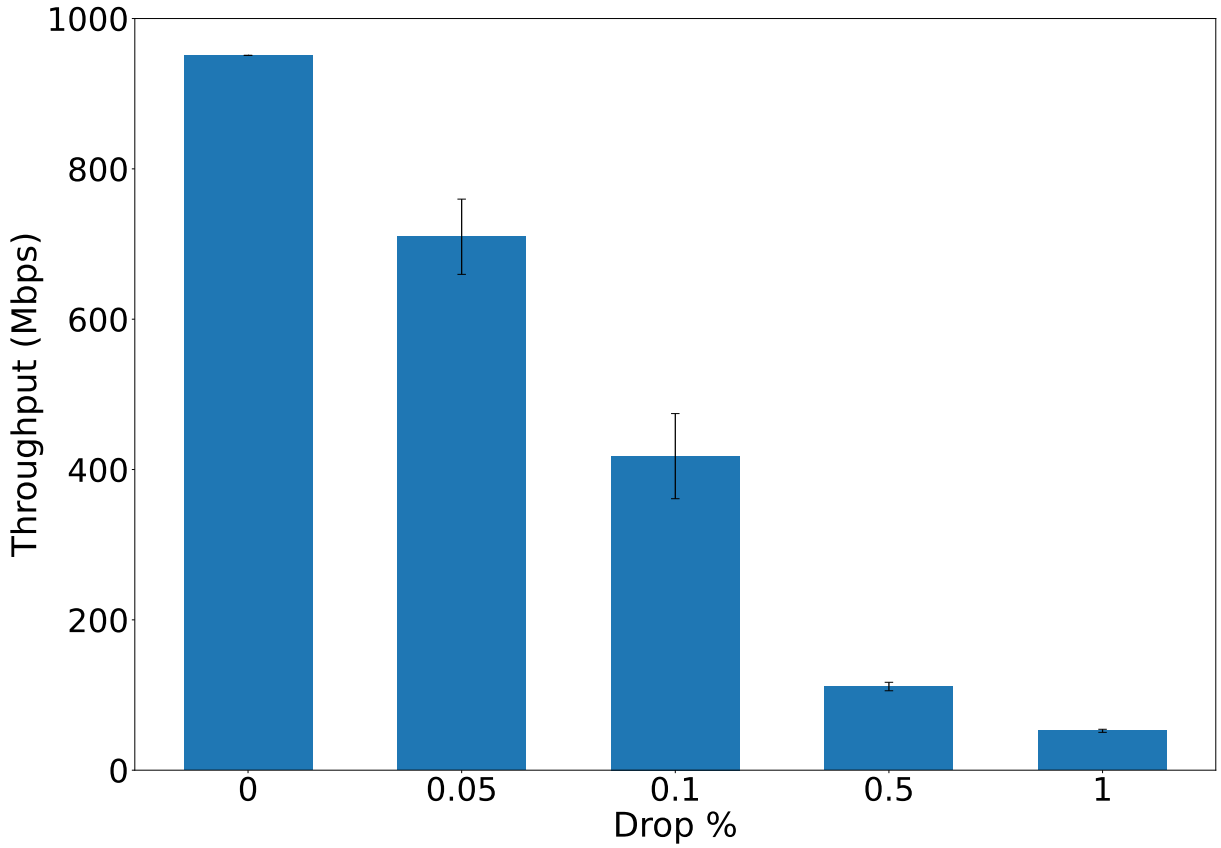



Figure 4.15: Average throughput for different drop rates at 5 ms RTT and 1 Gbps over the classical communication stack. Error bars show the standard deviation.

noticeably.

Figure 4.17 shows the average throughput for different percentages of jittered packets for 5 ms RTT. The results show a similar trend to Figure 4.16. Introducing an additional 10 ms to a small percentage of packets greatly affects throughput. These observations clearly demonstrate the communication stack's sensitivity to latency spikes.

### 4.2.3 Comparison Between Classical and 5G Communication Stacks

For the 1 ms case in Figure 4.14 and Figure 4.16, the behavior is similar. Throughput is stable between 0% and 0.1%, but declines beyond 0.5%. At 0.5% packet drop, there is no

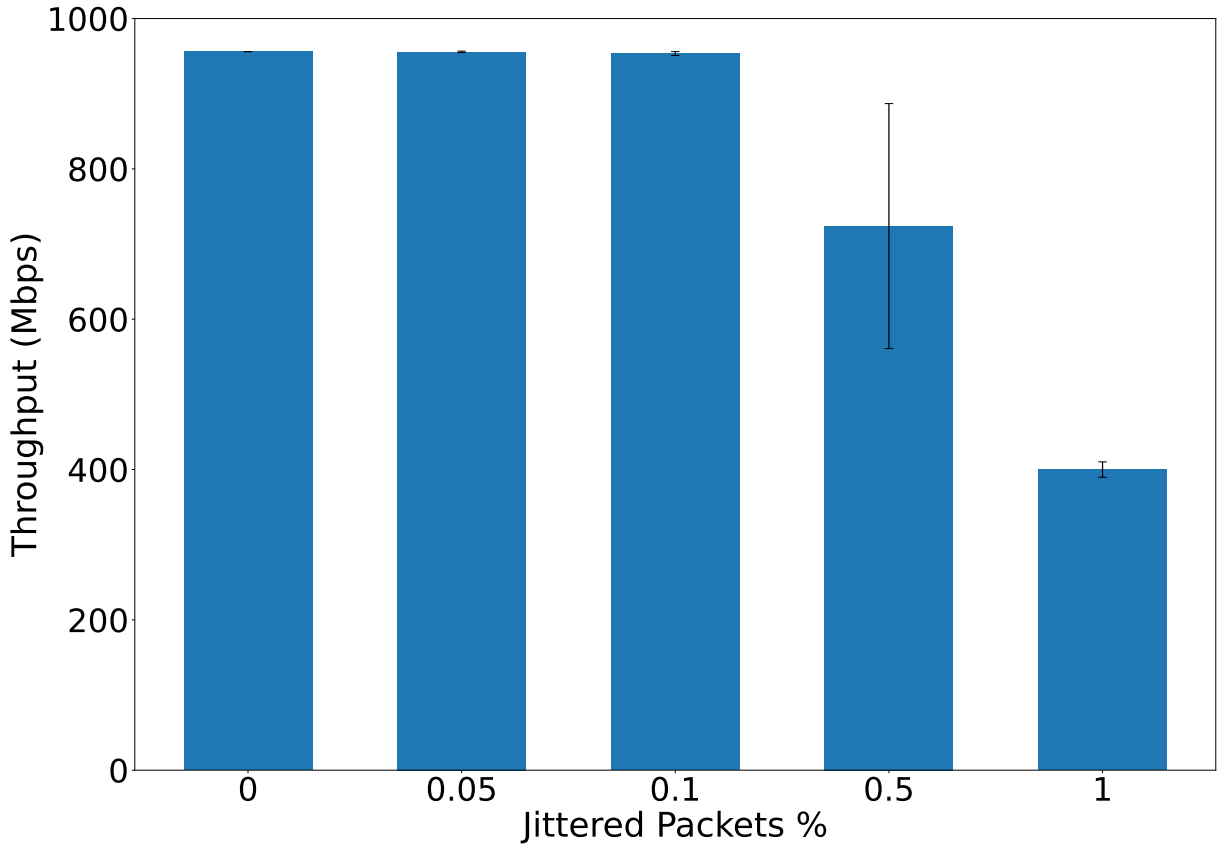



Figure 4.16: Average throughput for different percentages of jittered packets at 1 ms RTT and 1 Gbps over the 5G communication stack. Error bars show the standard deviation.

significant statistical difference in throughput for both the 5G and classical communication stacks, whereas at 1% packet drop, the classical communication stack achieves 15% higher throughput compared to the 5G communication stack.

For the 5 ms case in Figure 4.15 and Figure 4.17, both the classical and 5G communication stacks show a significant throughput decrease as the percentage of affected packets increases. With the classical communication stack, the decline is more immediate and pronounced, while for the 5G stack, the decline starts at 0.5%. Overall, the 5G communication stack achieves higher throughput at 5 ms RTT. The reason for this is that the additional 10 ms is 2 RTTs, while for the 1 ms case, it is more than 3 RTTs which causes TCP to reduce the congestion window, resulting in slowing down the transmission. Moreover, if acknowledgments are delayed, it is likely that the TCP client will timeout, causing more

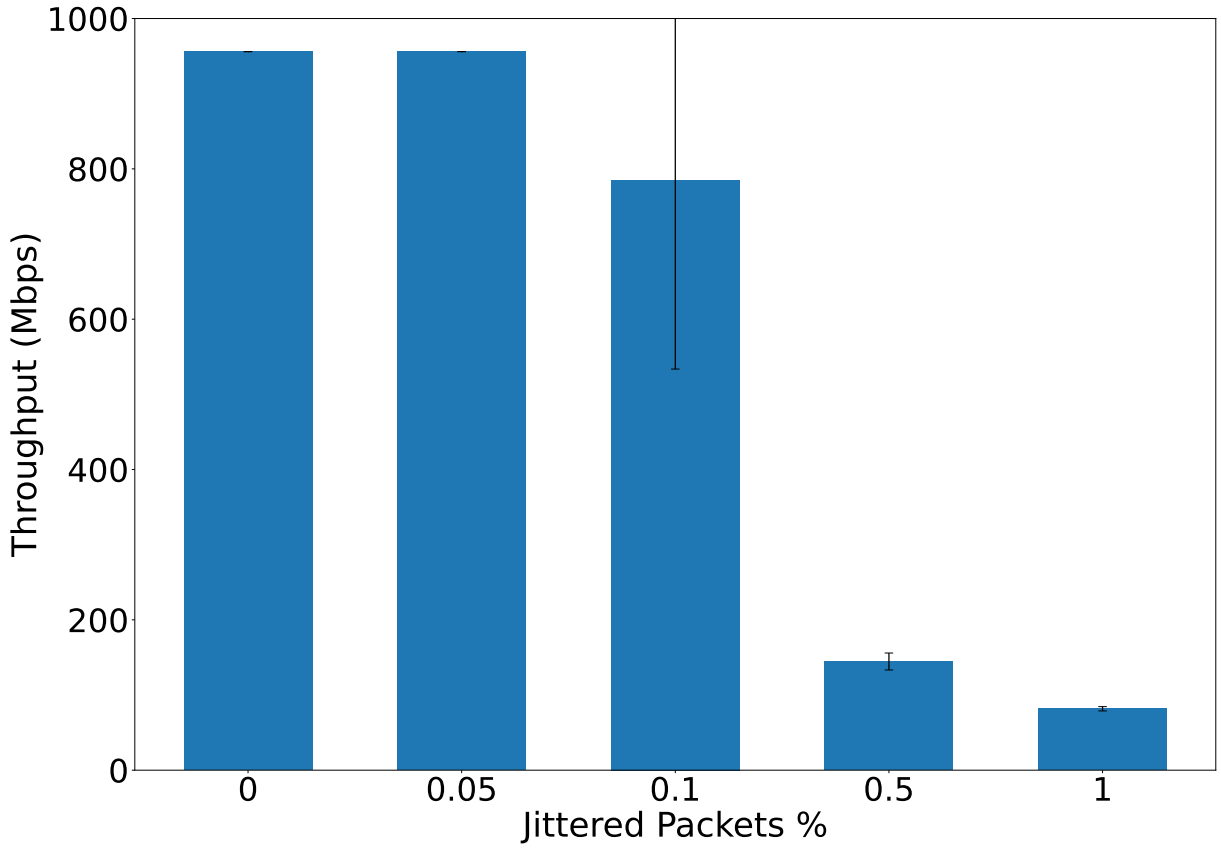



Figure 4.17: Average throughput for different percentages of jittered packets at 5 ms RTT and 1 Gbps over the 5G communication stack. Error bars show the standard deviation.

decreases in the congestion window.

### 4.2.4 Behavior of Congestion Control Protocols Over Classical and 5G Communication Stacks

To study the behavior of different congestion control protocols in this use case, I run experiments over both the classical and 5G communication stacks. Both experiments run on 1 ms RTT and 1 Gbps links. In the classical stack experiment, the packet loss is 0.5%, while in the 5G stack experiment, 10 ms jitter affects 0.5% of the packets. The congestion control protocols evaluated are BBR, LP, Westwood, CUBIC, Illinois, and DCTCP (Chapter 3).

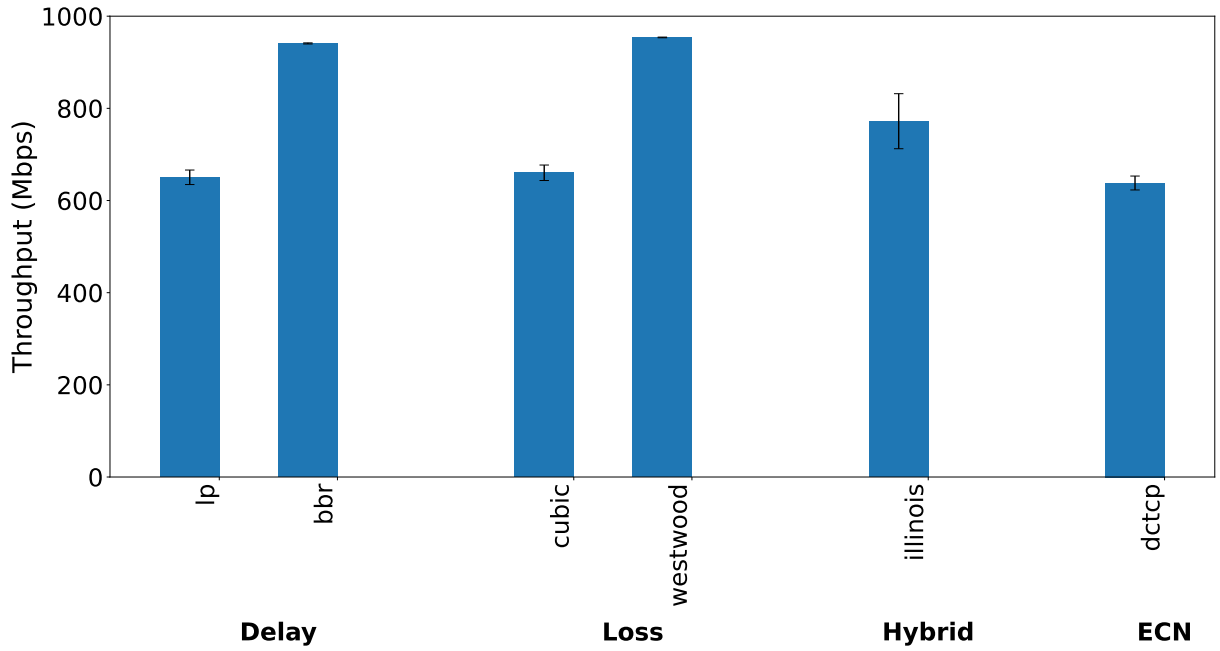



Figure 4.18: Average throughput for different congestion control protocols at 0.5% drop, 1 ms RTT, and 1 Gbps over the classical communication stack. Error bars show the standard deviation.

Figure 4.18 shows the average throughput for different congestion control protocols over the classical communication stack. The results vary across the different congestion protocols. Generally, the delay-based and the loss-based protocols have similar performance, with the loss-based protocols performing slightly better. Illinois, the hybrid protocol, outperforms CUBIC and LP but not Westwood and BBR. DCTCP achieves the lowest throughput at 638 Mbps, though its performance is still comparable to that of CUBIC and LP. Over the classical communication stack, the congestion control protocols utilize the following percentages of the link bandwidth:

- LP: 65%.
- BBR: 94%.
- CUBIC: 66%.
- Westwood: 95%.

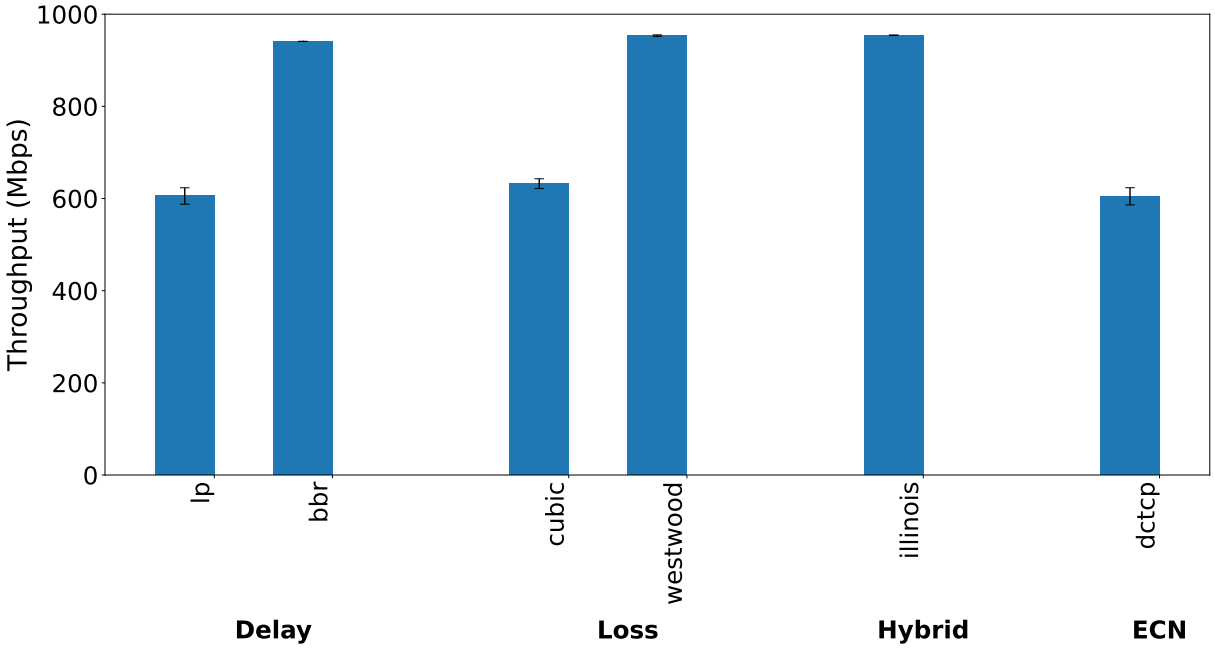



Figure 4.19: Average throughput for different congestion control protocols at 10ms jitter affecting 0.5% of packets, 1 ms RTT, and 1 Gbps over the 5G communication stack. Error bars show the standard deviation.

- Illinois: 77%.
- DCTCP: 64%.

Figure 4.19 shows the average throughput for different congestion control protocols over the 5G communication stack. The results show a similar trend to that observed in Figure 4.18. However, Illinois performs well in this setup, reaching 954.3 Mbps. Over the 5G communication stack, the congestion control protocols utilize the following percentages of the link bandwidth:

- LP: 61%.
- BBR: 94%.
- CUBIC: 63%.

- Westwood: 95%.
- Illinois: 95%.
- DCTCP: 60%.

### 4.2.5 Summary

Packet drop impacts throughput over both the classical and 5G communication stacks. At 1 ms RTT, throughput for both stacks is similar for drop rates less than 1%. At 1% packet drop rate, the classical communication stack outperforms the 5G communication stack. At 5 ms RTT, the 5G communication stack achieves higher overall throughput. The reason for this is that the additional 10 ms is 2× the average RTT, while for the 1 ms case, it is more than 3× RTT which causes TCP to reduce the congestion window, resulting in slowing down the transmission. Moreover, if acknowledgments are delayed, it is likely that the TCP client will timeout, causing more decreases in the congestion window.

Congestion control protocols perform similarly in both the classical and 5G communication stack. The only exception is Illinois, which achieves 23.6% more throughput over the 5G communication stack.

## 4.3 Use Case III: Edge as an Accelerator for WAN Communication

Edge clusters can accelerate TCP communications over wide-area networks through TCP splitting. Figure 4.20 shows this use case. The data center hosts the service, and the edge is in the middle between the client and the server, running a TCP proxy. The end-to-end latency between the data center and the end user is 150 ms. With the edge splitting the connection, the sub-connections have equal characteristics, with a 75 ms RTT. I study this use case using the classical communication stack, in which packet drops result in packet loss within TCP streams.

To explore the benefits of TCP splitting in this context, I address the following key questions:

- What is the impact of TCP proxy placement on end-to-end throughput?
- What is the impact of packet drop on end-to-end throughput?

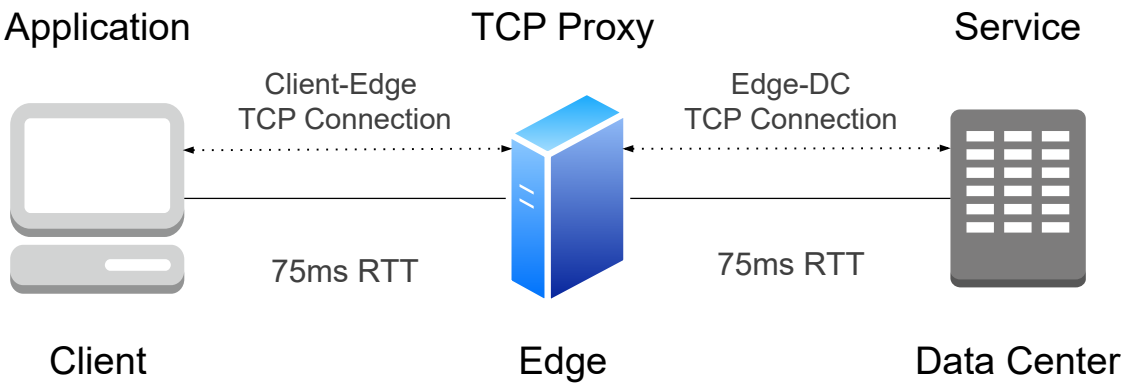



Figure 4.20: Edge as an accelerator for WAN communication.

- How do different congestion control protocols perform?
- What is the impact of TCP buffer sizes on end-to-end throughput? Are the default buffer values enough?
- Under what traffic conditions does TCP splitting enhance the performance? For instance, at what flow size does TCP splitting bring performance benefits?

Unless otherwise specified, I conduct the experiments on a 1 Gbps link with the maximum TCP buffer configuration, using TCP CUBIC.

### 4.3.1 Impact of Proxy Placement

To study the impact of proxy placement in this context, I conduct an experiment on a 1 Gbps link, with 150 ms RTT and a packet drop rate of 0.5%. The placement of the proxy varies from 10 ms to 140 ms RTT from the client. With the presence of the TCP proxy, I divide the original drop rate proportionally between the client and server segments based on the RTT of each segment.

Figure 4.21 shows the average end-to-end throughput for different proxy placements. The results show that the proxy placement affects the end-to-end throughput. Throughput increases as the proxy placement gets closer to the middle point, reaching its highest at 75 ms from the client. As the placement gets further, throughput decreases until there is no statistical difference between the no proxy and proxy cases, when the proxy is at around 130 ms from the client.

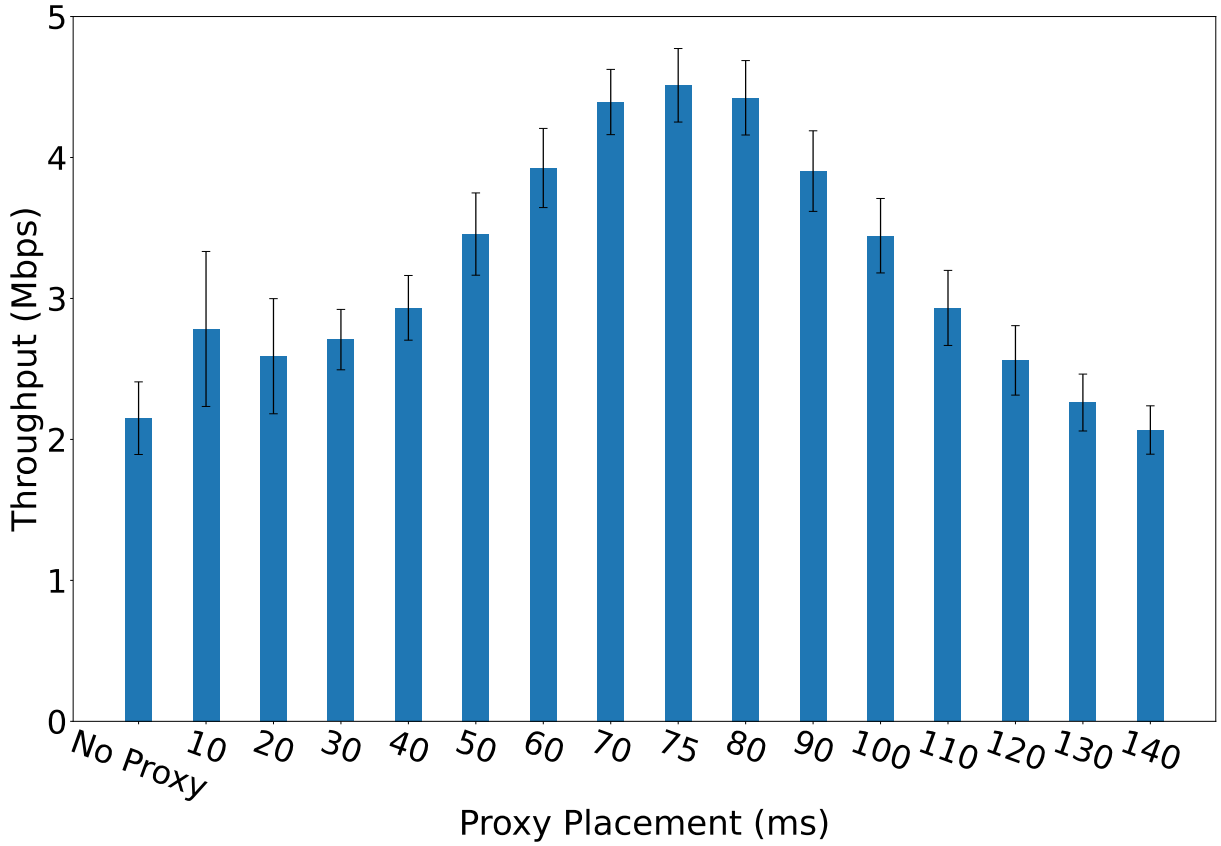



Figure 4.21: Average throughput for different proxy placements at 150 ms RTT, 1 Gbps, and 0.5% packet drop. Error bars show the standard deviation.

For the remainder of this section, I place the proxy in the middle between the client and the server, as it results in the highest increase in throughput.

### 4.3.2 Impact of Packet Drop

I conduct an experiment to study the impact of packet drop on the end-to-end throughput in this context. The experiment runs on 1 Gbps link with 150 ms RTT. I place a TCP proxy at 75 ms from the client, and vary the drop rate between 0%, 0.05%, 0.1%, 0.5%, and 1%. As the proxy sits in the middle between the client and the server, it divides the original TCP connection into two equal sub-connections. At 1% packet drop, each sub-connection will have a packet drop rate of 0.5% and so on.

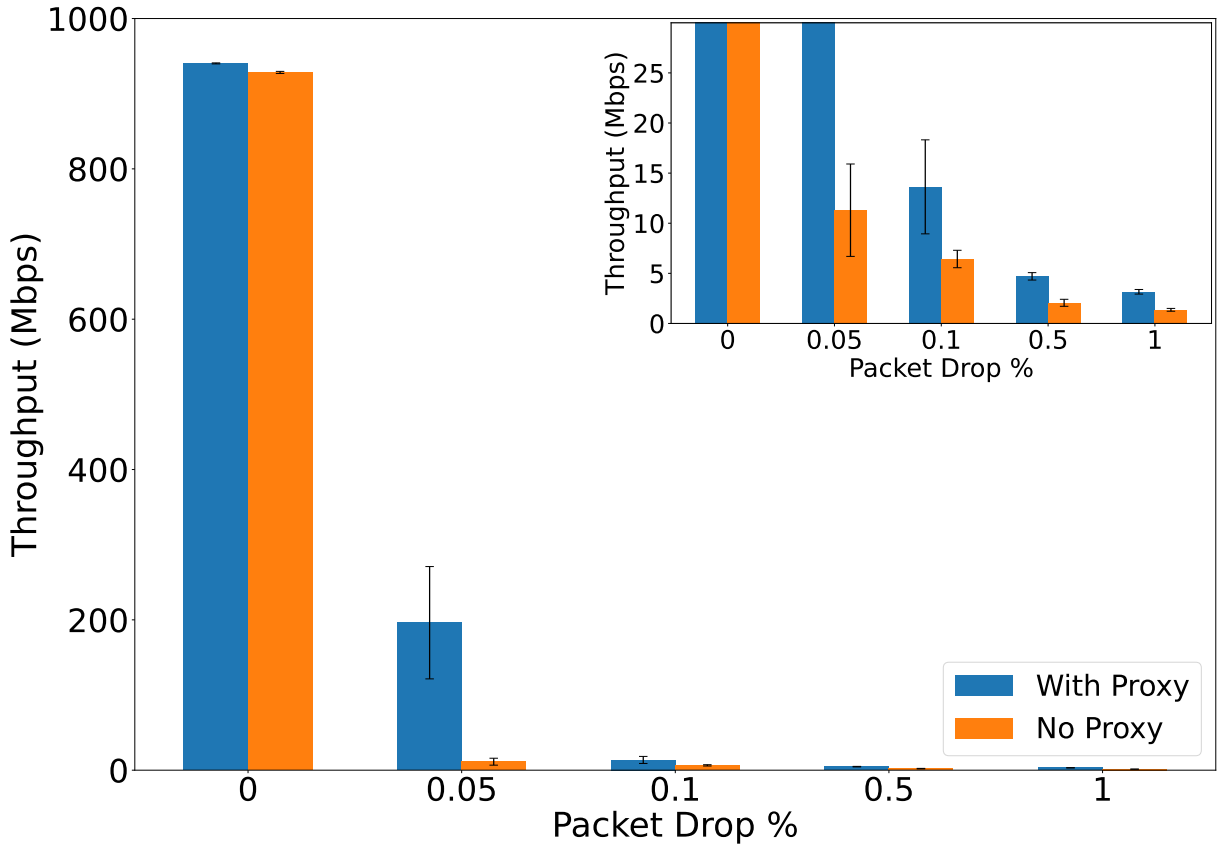



Figure 4.22: Average throughput for different drop rates at 150 ms RTT, 1 Gbps, and edge at 75 ms. Error bars show the standard deviation. The smaller figure in the top right corner provides a zoomed-in view of the lower bars.

Figure 4.22 shows the average throughput for different drop rates. Throughput decreases as the drop rate increases, with the 0% drop case having the best end-to-end throughput, as expected. At 0% packet drop, splitting the connection is not useful, causing only a 1.3% increase in throughput. As the drop rate increases, TCP splitting shows great performance improvement, resulting in 17.3×, 2.1×, 2.3×, and 2.3× increases in throughput for 0.05%, 0.1%, 0.5%, and 1% packet drop, respectively. This is because with TCP splitting, each sub-connection has a lower RTT, resulting in faster recovery as the feed-back loop of the TCP connection is shortened.

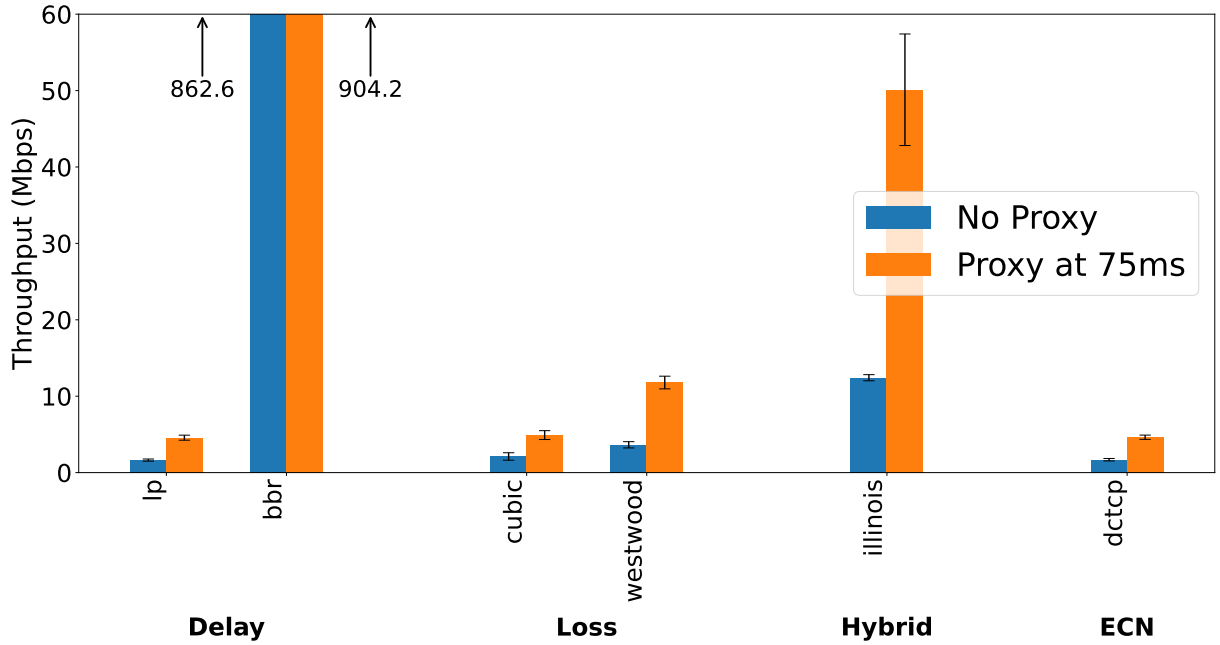



Figure 4.23: Average throughput for different congestion control protocols at 150 ms RTT, 1 Gbps, 0.5% drop, and edge at 75 ms. Error bars show the standard deviation.

### 4.3.3 Behavior of Congestion Control Protocols

I study the behavior of different congestion control protocols in this context by running an experiment on a 1 Gbps link with 150 ms RTT and 0.5% packet drop. I deploy a TCP proxy at 75 ms RTT from the client. The congestion control protocols evaluated are BBR, LP, Westwood, CUBIC, Illinois, and DCTCP (Chapter 3).

Figure 4.23 shows the average end-to-end throughput for different congestion control protocols. The figure shows that that every protocol, other than BBR, benefits significantly from employing TCP splitting. Using a TCP proxy causes increases of 2.3×, 3.2×, 2.7×, 4×, 2.7×, and 4.8% for CUBIC, Westwood, LP, Illinois, DCTCP, and BBR, respectively. With TCP splitting, the second best protocol is Illinois at 50.11 Mbps, followed by Westwood at 11.79 Mbps, while CUBIC, LP, and DCTCP achieve comparable results.

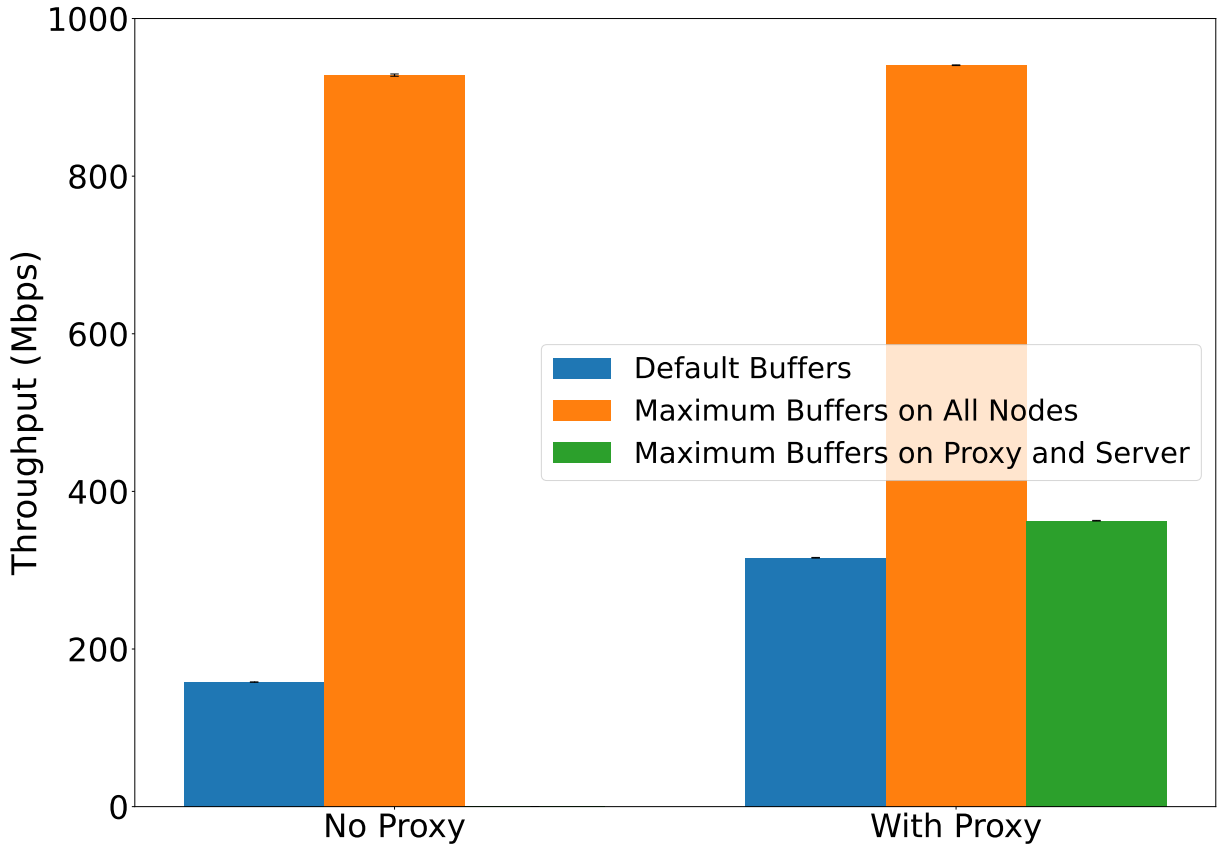



Figure 4.24: Average throughput for different buffer configurations at 150 ms RTT, 1 Gbps, and edge at 75 ms. Error bars show the standard deviation.

### 4.3.4 Impact of TCP Buffer Sizes

Buffer sizes substantially affect TCP performance, and the default configurations for TCP buffers in Linux are normally too small to effectively utilize the network capabilities for connections with large RTT. I conduct two experiments to examine the impact of buffer sizes on the end-to-end throughput in this use case. The first experiment runs on a 1 Gbps link with 150 ms RTT, and no packet drop, while the other runs with the same configuration plus a 0.5% packet drop (0.25% on each sub-connection). For both experiments, I deploy a TCP proxy at 75 ms from the client.

Figure 4.24 shows the average throughput for different buffer configurations without packet loss. As expected, with default TCP buffers, the network is barely utilized, with

the no proxy throughput capping at 158 Mbps. With the proxy, throughput increases by 2×. The proxy causes a significant improvement in performance, but still, throughput caps at 315.8 Mbps, which is still nowhere near the 1 Gbps capacity of the link, considering there are no packet losses to cause performance degradation. With maximum buffers, the throughput reaches 940.75 Mbps with a proxy and 928.2 Mbps without a proxy. With maximum buffers, the TCP proxy does not cause a significant increase in throughput. Moreover, increasing the buffer sizes on the proxy-server link causes a 2.3× throughput increase compared to the default buffers no proxy case. Even though this is a significant improvement, client buffers are still limiting the performance, with the throughput reaching only 362.8 Mbps.

It is important to note that TCP splitting causes significant performance enhancements, especially when it is not possible to modify the buffers on the client side. With maximum buffers, link saturation is possible, with the throughput reaching 940.4 Mbps.

Figure 4.25 shows the average end-to-end throughput for different buffer configurations with 0.5% packet drop. The results clearly demonstrate that using the proxy boosts the end-to-end throughput. With default buffers, performance improves by 61.2%. With maximum buffers, TCP splitting increases performance by 2.2×. Throughput with proxy, whether it is with default buffers, maximum buffers on all nodes, or maximum buffers on the proxy and the sever only, are equal. Using maximum buffer sizes with the no-proxy case resulted in a 29% decrease in throughput compared to the no-proxy case with default buffers. This is due to packet drops, which, as I investigated, prevent full buffer utilization by triggering retransmissions and reductions in the congestion window.

### 4.3.5 Impact of TCP Flow Size

TCP splitting significantly improves performance over the classical communication stack when transmitting large amounts of data. In this section, I investigate the flow sizes at which TCP splitting is beneficial.

I conduct an experiment to measure the time from the moment a TCP socket is opened until the socket is closed successfully, this includes the time to send the data over the network. I repeatedly measure this time for different data sizes. For each size, I run the experiment for at least 60 seconds or until I send the data at least 10 times, then I average these measurements. I refer to this average as the response time. I measure the response time because smaller flow sizes can not fully utilize the network capacity.

I run the experiment on a 1 Gbps link with 150 ms RTT and 0.5% packet drop. Flow sizes vary from 1 KB to 1 GB, with placing the proxy at 75 ms from the client.

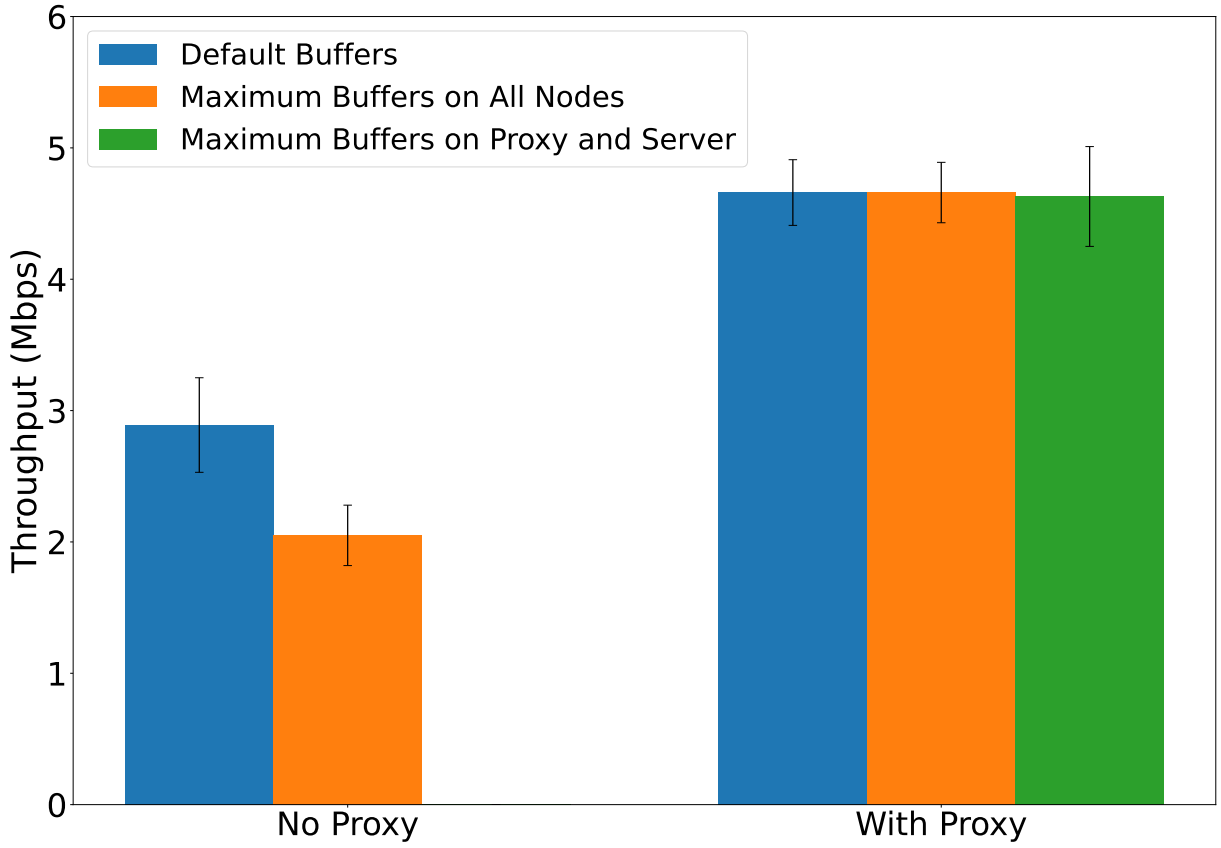



Figure 4.25: Average throughput for different buffer configurations at 150 ms RTT, 1 Gbps, 0.5% packet drop, and edge at 75 ms. Error bars show the standard deviation.

Figure 4.26 shows the average response time for different data sizes over the classical communication stack. Response times are always lower with the TCP proxy. For data sizes from 1 KB - 64 KB, response times remain constant for the three setups (no proxy, with proxy, and no proxy no drop). After that, as the size increases, the response time increases. Response times for the no proxy case are 2.3× more than those for the proxy setup.

### 4.3.6 Summary

TCP proxy placement affects the end-to-end throughput. Throughput is highest when the proxy is positioned in the middle between the client and the server in terms of RTT. TCP

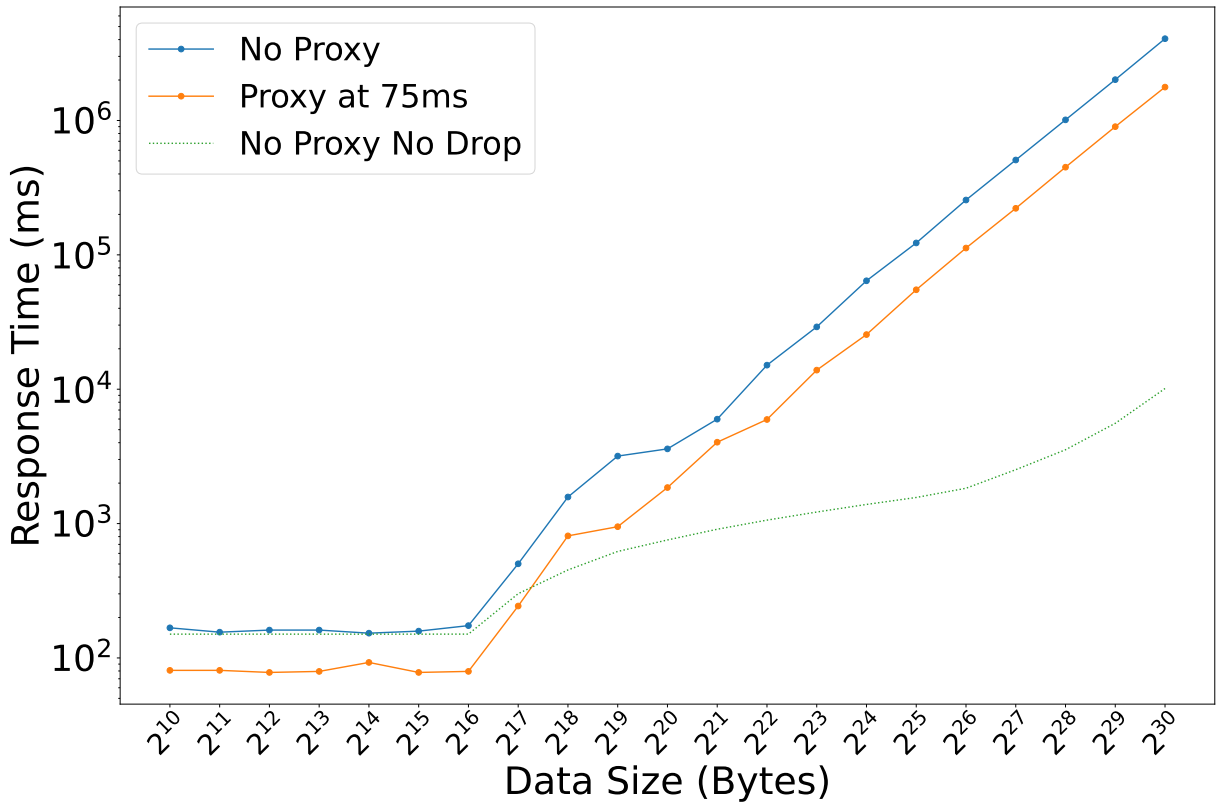



Figure 4.26: Average response time for different flow sizes at 150 ms RTT, 1 Gbps, and 0.5% drop.

splitting is beneficial over lossy connections as it reduces the RTT for each sub-connection, leading to faster recovery due to the shortened feed-back loop.

Congestion control protocols perform differently. However, all congestion protocols benefit greatly from splitting the TCP connection, except for BBR. TCP buffer sizes affect performance; without packet drop, increasing TCP buffer sizes significantly enhances the performance. However, packet drops over larger RTTs prevent full buffer utilization by triggering retransmissions and reductions in the congestion window. As a result, even default buffer sizes may not be fully utilized, meaning that increasing buffer sizes might not enhance throughput significantly. Splitting the TCP connection brings performance benefits with flow sizes over 64 KB. With flow sizes of 64 KB or less, the response time is low because the client-to-proxy connection is shorter than the original no-proxy TCP connection.

# Chapter 5

# Related Work

In this section, I present a survey of literature related to this work, including research efforts on edge computing, content delivery networks, TCP splitting, theoretical TCP models, and congestion control.

## 5.1 Edge Applications

Edge computing is used in a wide range of applications across various industries, for their real-time data processing and low-latency services. These applications include real-time object detection [46], smart healthcare [83], real-time facial expression recognition [86], connected autonomous vehicles [78, 73], and VR video streaming [48]. Typically, these applications employ a distributed architecture, where data processing is done closer to its source (at the edge). These applications focus on leveraging edge computing to offload parts of the application to a close by edge cluster. They use standard TCP-based communication. In this thesis, I study using edge computing to accelerate application-agnostic TCP and UDP communication.

## 5.2 Content Delivery Networks

One early usage of edge computing is to support content delivery through Content Delivery Networks (CDNs). CDNs consist of a network of geographically distributed servers designed to improve the performance of web content delivery by caching static content

closer to users. When a client requests content, the request is redirected to the closest CDN server, which responds with the cached data, reducing latency and the load on the original server. Examples of CDN services are Cloudflare [16], Akamai [5], and Google Cloud CDN [26].

Several studies illustrated the notable impact of content delivery networks on enhancing web access performance. Vakali and Pallis [81] provide an overview of the CDN architecture and common CDN service providers. They highlight the role of CDNs in reducing bandwidth consumption and latency by caching content closer to users, ensuring faster delivery during high-demand periods. Krishnamurthy et al. [42] conducted a study on CDN companies that employ DNS redirection and URL rewriting for load balancing. Moreover, Saroiu et al. [68] analyze content delivery for four different network traffic patterns, including HTTP web traffic, traffic from Akamai and Kazaa, and peer-to-peer file sharing traffic from Gnutella. They highlight the increasing importance of new content deliver systems.

CDN placement is an important factor that influences the performance and efficiency of content delivery systems. Several algorithms have been developed. Sahoo et al. [66] present a comprehensive survey of CDN placement algorithms. The study summarizes and categorizes replica server placement algorithms for two types of CDNs: traditional and emerging paradigm based CDN. Traditional CDNs use conventional web technologies to build replicas, while more recent CDNs leverage emerging paradigms such as cloud computing and Network Functions Virtualization. Placement algorithms for traditional CDNs can be categorized as algorithms that are: QoS aware, data consistency aware, energy aware, and others. Placement algorithms vary in their optimization objectives, such as minimizing one or more of the following: latency [37], hop count [76], network traffic [82], cost [33], and energy consumption [54].

Unlike previous work, this work studies using the edge for accelerating end-to-end communication and to understand the performance of communication for edge applications. I examine the viability of edge computing for improving end-to-end performance in WANs and 5G networks. Although they might sound similar to the focus of this research, TCP proxies operate differently. TCP splitting can accelerate client-to-server (uplink) communication as well as server-to-client (downlink) communication, and the content is not necessarily static. Unlike previous studies, I study a different class of applications in which I use TCP splitting to upload data from end devices to the cloud.

## 5.3 TCP Splitting

Previous efforts study the impact of TCP splitting on communication performance. Kopparty et al. [41] highlight how TCP's throughput and fairness suffer in mobile ad hoc networks, and investigate the impact of splitting long TCP connections to improve fairness.

Maki et al. [52] analyze the throughput of TCP overlay networks. The paper states that some undesirable interactions between the split TCP connections could lead to performance degradation. The paper introduces an approach to estimate the overall throughput when using a TCP proxy, while taking into account the possible performance degradation. The results illustrate how effective the TCP proxy mechanism is. The paper states that the performance degradation becomes quite significant when the network is congested. Moreover, the authors claim that their approach is independent of the TCP congestion control protocol used.

Yamanegi et al. [85] conducted experiments using TCP splitting on a public network between Osaka and Tokyo. The use of a proxy is compared to the no proxy case. The paper includes an experiment in which they change the TCP variant on one link from Reno to High-speed TCP [21]. It also states that the effectiveness of a TCP proxy depends on its placement within the network. Yamanegi et al. [85] and Maki et al. [52] mention that TCP splitting is most effective when that proxy is placed such that the delay between the client and the server is divided equally.

Maki et al. [53] analyze the TCP proxy mechanism. They state that TCP splitting can cause performance degradation, especially when the network congestion along the split TCP connections is small. They minimize the performance degradation by increasing the send buffer size of the sub-connection, and the required size is significantly larger than for a standard end-to-end TCP connection.

There is limited research effort on TCP splitting in 5G networks. Hayes et al. [30] highlight the importance of TCP splitting for 5G communications. Their results indicate that TCP proxies will be more important for 5G than 4G LTE networks. Unlike these efforts, I conduct a deep empirical evaluation of TCP splitting and show that TCP splitting is not always beneficial in a 5G environment, especially when considering the HARQ protocol and sender scheduling. Kim et al. [39] study the performance of TCP splitting in 4G LTE and 3G wireless networks.

Unlike previous efforts, this work is the only study that investigates the impact of the HARQ protocol and the 5G frame structure on the end-to-end communication performance.

Moreover, this study investigates TCP splitting across a significantly larger number of congestion control protocols compared to previous research efforts.

## 5.4 Theoretical TCP Models

Instead of using an empirical approach to study the impact of TCP splitting, I considered using an analytical TCP model. Unfortunately, there are no accurate models for many congestion protocols I use in my study including TCP CUBIC, the default congestion control protocol in Linux.

There is a lot of research on modeling TCP for better comprehension of its behavior and performance in different scenarios. It is particularly beneficial for theoretical analysis and performance prediction without the need for complex real-world testing. There exists established models for TCP congestion control algorithms, such as the Mathis formula (also known as the TCP Macroscopic Model) [57] for TCP Reno. However, theoretical studies on TCP CUBIC are limited.

The aforementioned models rely heavily on certain assumptions, making them more suitable for pristine and controlled environments, but less accurate in real-world, dynamic conditions. In other words, these models facilitate throughput prediction under ideal conditions.

I applied the Mathis model to several configurations and compared the actual throughput values with the theoretical numbers. The model failed to provide accurate throughput predictions. In fact, the average error rate between the theoretical and expected throughput was 119.5%. Mathis model contains a constant of proportionality C, that is determined by a combination of the TCP implementation, acknowledgment strategy, and loss mechanism. To get a good estimate of C, I ran experiments and used the formal to find C. The results show high variance between the calculated C's for each configuration, meaning that it is not possible to have one constant C that would work for all of the configurations. Mathis, the main author of the Mathis model paper [57], acknowledged the model's limitations. In a recent article [56], Mathis notes that "The TCP Macroscopic Model will be completely obsolete soon...[it] requires relatively large buffers to function as intended, while Moore's law is making them uneconomical". He states that the emergence of TCP BBR renders the model irrelevant, and challenges three decades of network research based on it.

Although there is no widely established model for TCP CUBIC, Poojary et. al [62] proposed a model that approximates the average congestion window size for long-lived CUBIC flows. To evaluate its accuracy, I conducted a series of experiments to record the

window size over time using the same packet size used in the study by Poojary et. al [62]. When comparing the average window size from the experiments with the theoretical output from their model, there was a significant difference between the theoretical predictions and the experimental data. The actual window size is significantly larger than the theoretically expected one. I have also experimented with setting the initial congestion window to 1 maximum segment size. This adjustment resulted in reducing the discrepancy, but the experimental results remained larger than predicted, with the error rate ranging between 33.2% to 68.9%.

## 5.5 Congestion Control Surveys

There is a humbling amount of research on congestion control protocols. There are surveys of common congestion control protocols used in Wireless Sensor Networks [22, 38, 70], which are a special type of wireless and ad hoc networks with unique properties and limitations. Al-Saadi et al. [6] survey delay-based and hybrid congestion control algorithms. Kushwaha et al. [43] examine router-based and source-based congestion control in high-speed wired networks. Rajesh et al. [65] provide a systematic review of congestion control techniques in ad hoc networks.

There are a number of studies that explore congestion protocols in edge environments. Xiao et al.[84] study multi-stage resource-aware congestion algorithms in edge computing environments. Resource-aware congestion control attempts to manage congestion by considering the available network resources and their utilization. Sakashita et al. [67] benchmark congestion protocols such as Reno, CUBIC and BBR in architectures that combine the concepts of both Cloud and Edge computing. Sivaraman et al. [75] study the learnability of congestion protocols, that is the ability of a protocol to function effectively in uncertain network conditions. Chaudhary et al. [14] offer a comparative study of different TCP variants in wireless networks.

There is little work that investigates various congestion protocols in the 5G environment. Lorincz et al. [49] investigate recent TCP variants in 5G networks, concentrating on the sub-6 GHz and mmWave spectrum. Haile et al. [28] discuss end-to-end congestion techniques in 4G/5G networks, with a brief mention of HARQ as a challenge. Hindawi et al. [32] review congestion protocols in mmWave networks while highlighting network errors and retransmission issues.

This work differs from previous studies by providing a comprehensive evaluation of congestion control algorithms in a 5G environment. It specifically focuses on the impact of

the HARQ protocol, the 5G frame structure, and resource allocation, which have not been studied before. Many of these studies lack practical evaluations, while my work follows an experimental approach to provide a thorough understanding of the behavior of these algorithms under different network conditions.

# Chapter 6

# Conclusion

I explore the viability of using edge clusters to optimize end-to-end communication performance in both WANs and 5G environments. The study focuses on three scenarios: accelerating TCP communications through TCP splitting in 5G deployments, hosting an entire application-level service or the latency-sensitive part of an application on an edge cluster, and deploying a TCP splitting service on edge clusters to support WAN communication. I explore these scenarios by following an empirical approach using a cluster of three nodes with network emulation to mimic the aforementioned use cases. I validate the emulation setup, configure TCP buffer sizes, and select representative congestion control protocols. I then run experiments with varying packet drop rates, communication stacks, congestion control protocols, and TCP buffer sizes.

I show that edge computing, especially through TCP splitting, can significantly improve end-to-end communication performance over the classical communication stack. This is because TCP splitting reduces the impact of packet drop by shortening the feedback loop of the TCP connection. TCP splitting over the 5G communication stack does not bring benefits and can reduce throughput. This is because the 5G communication stack causes latency spikes as it hides packet drop from the TCP layer. With the edge being close to the client, this latency spike is significant compared to the link RTT, resulting in reducing the congestion window and reducing throughput. Furthermore, over the classical communication stack, TCP splitting brings higher benefit for flows larger than 64 KB.

These findings provide valuable insights into how edge clusters can accelerate TCP communication in different network environments. I identify two timely and high-impact research ideas. Future work could explore alternative communication stacks for MEC deployments that are close to the client, as the 5G communication stack does not always

achieve the best performance. This includes investigating modifications at the link layer, using a different congestion control protocol, or developing a co-design approach between the two. A second idea is related to optimizing resource usage for TCP splitting. TCP splitting uses edge resources but it does not bring benefits to all TCP flows. My study shows that TCP splitting is beneficial for certain flow sizes, which can be easily measured after transmission. It is not clear how much data will be transmitted over a TCP connection at the moment it is opened. An interesting future research problem is to design a prediction technique based on the user's history to decide whether to use a TCP splitting service at the edge for this user.

# References


[1] 3rd Generation Partnership Project (3GPP). 5G; NR; Physical channels and modulation. Technical Report TS 38.211 version 18.2.0 Release 18, 3GPP, 2024. Accessed: 2024-07-18.

[2] 3rd Generation Partnership Project (3GPP). Nr; nr and ng-ran overall description; stage-2. Technical Report TS 38.300 version 18.1.0, ETSI, 2024.

[3] 3rd Generation Partnership Project (3GPP). System architecture for the 5g system (5gs). Technical Report TS 23.501 version 18.5.0, ETSI, 2024.

[4] Sassan Ahmadi. *5G NR : architecture, technology, implementation, and operation of 3GPP New Radio standards*. Academic Press is an imprint of Elsevier, London, United Kingdom, 1st edition edition, 2019.

[5] Akamai. Akamai. https://www.akamai.com/, 2024. Accessed: 2024-08-07.

[6] Rasool Al-Saadi, Grenville Armitage, Jason But, and Philip Branch. A survey of delay-based and hybrid tcp congestion control algorithms. *IEEE Communications Surveys & Tutorials*, 21(4):3609–3638, 2019.

[7] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[8] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP congestion control. RFC 5681 (Standard), September 2009. Accessed: 2024-08-07.

[9] Eneko Atxutegi, Fidel Liberal, Habtegebreil Kassaye Haile, Karl-Johan Grinnemo, Anna Brunstrom, and Ake Arvidsson. On the use of tcp bbr in cellular networks. *IEEE Communications Magazine*, 56(3):172–179, 2018.

[10] Zhe Bing, Xing Wang, Zhenliang Dong, Luobing Dong, and Tao He. A novel edge computing architecture for intelligent coal mining system. *Wireless Networks*, 29(4):1545–1554, 2023.

[11] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020.

[12] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. When to use and when not to use bbr: An empirical analysis and evaluation study. In *Proceedings of the Internet Measurement Conference*, pages 130–136, 2019.

[13] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

[14] Pooja Chaudhary and Sachin Kumar. Comparative study of tcp variants for congestion control in wireless network. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 641–646, 2017.

[15] Federico Chiariotti, Andrea Zanella, Stepan Kucera, and Holger Claussen. Bbr-s: A low-latency bbr modification for fast-varying connections. *IEEE Access*, 9:76364–76378, 2021.

[16] Cloudflare. Cloudflare. https://www.cloudflare.com/, 2024. Accessed: 2024-08-07.

[17] CloudLab. CloudLab: A cloud testbed for research on the future internet. https://www.cloudlab.us/. Accessed: 2024-08-1.

[18] A. Cohen and R. Cohen. A dynamic approach for efficient tcp buffer allocation. *IEEE Transactions on Computers*, 51(3):303–312, 2002.

[19] R. Comroe and D. Costello. Arq schemes for data transmission in mobile radio systems. *IEEE Journal on Selected Areas in Communications*, 2(4):472–481, 1984.

[20] ETSI. Nr; medium access control (mac) protocol specification. Technical Report TS 38.321 version 17.0.0, ETSI, 2022.

[21] Sally Floyd. Highspeed tcp for large congestion windows, December 2003.

[22] Ali Ghaffari. Congestion control mechanisms in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, 52:101–115, 2015.

[23] Google. BBR. https://github.com/google/bbr. Accessed: 2024-11-05.

[24] Google. BBRv2. https://datatracker.ietf.org/meeting/106/materials/slides-106-iccrg-update-on-bbrv2. Accessed: 2024-11-06.

[25] Google. BBRv3. https://datatracker.ietf.org/meeting/119/materials/slides-119-ccwg-bbrv3-overview-and-google-deployment-00. Accessed: 2024-11-06.

[26] Google Cloud. Google cloud cdn, 2024. Accessed: 2024-09-20.

[27] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[28] Habtegebreil Haile, Karl-Johan Grinnemo, Simone Ferlin, Per Hurtig, and Anna Brunstrom. End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks. *Computer Networks*, 186:107692, 2021.

[29] Samira Hayat, Roland Jung, Hermann Hellwagner, Christian Bettstetter, Driton Emini, and Dominik Schnieders. Edge computing in 5g for drone navigation: What to offload? *IEEE Robotics and Automation Letters*, 6(2):2571–2578, 2021.

[30] David A. Hayes, David Ros, and Özgü Alay. On the importance of tcp splitting proxies for future 5g mmwave communications. In *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, pages 108–116, 2019.

[31] Belghol Hibat Allah and Idrissi Abdellah. Mec towards 5g: A survey of concepts, use cases, location tradeoffs. *Transactions on Machine Learning and Artificial Intelligence*, 5(4), 2017.

[32] Bashar Hindawi and Ahmed S. Abbas. Congestion control techniques in 5g mm wave networks: A review. In *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*, pages 305–310, 2021.

[33] Kin-Hon Ho, Stylianos Georgoulas, Mina Amin, and George Pavlou. Managing traffic demand uncertainty in replica server placement with robust optimization. In *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems: 5th International IFIP-TC6 Networking Conference, Coimbra, Portugal, May 15-19, 2006. Proceedings 5*, pages 727–739. Springer, 2006.

[34] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th international conference on network protocols (ICNP)*, pages 1–10. IEEE, 2017.

[35] Peng Huang, Liekang Zeng, Xu Chen, Ke Luo, Zhi Zhou, and Shuai Yu. Edge robotics: Edge-computing-accelerated multirobot simultaneous localization and mapping. *IEEE Internet of Things Journal*, 9(15):14087–14102, 2022.

[36] iperf Team. *iperf - The TCP, UDP, and SCTP Network Bandwidth Measurement Tool*, 2023. https://iperf.fr/iperf-doc.php.

[37] Sugih Jamin, Cheng Jin, Anthony R Kurc, Danny Raz, and Yuval Shavitt. Constrained mirror placement on the internet. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 1, pages 31–40. IEEE, 2001.

[38] Mohamed Amine Kafi, Djamel Djenouri, Jalel Ben-Othman, and Nadjib Badache. Congestion control protocols in wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 16(3):1369–1390, 2014.

[39] Bong Ho Kim and Doru Calin. On the split-tcp performance over real 4g lte and 3g wireless networks. *IEEE Communications Magazine*, 55(4):124–131, 2017.

[40] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *ICC 1979; International Conference on Communications, Volume 3*, volume 3, pages 43–1, 1979.

[41] Swastik Kopparty, Srikanth V Krishnamurthy, Michalis Faloutsos, and Satish K Tripathi. Split tcp for mobile ad hoc networks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 1, pages 138–142. IEEE, 2002.

[42] Balachander Krishnamurthy, Craig Wills, and Yin Zhang. On the use and performance of content distribution networks. IMW '01, page 169–182, New York, NY, USA, 2001. Association for Computing Machinery.

[43] Vandana Kushwaha and Ratneshwer Gupta. Congestion control for high-speed wired network: A systematic literature review. *Journal of Network and Computer Applications*, 45:62–78, 2014.

[44] A. Kuzmanovic and E.W. Knightly. Tcp-lp: a distributed algorithm for low priority data transfer. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 3, pages 1691–1701 vol.3, 2003.

[45] Linux Foundation. Linux traffic control. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: 2024-08-1.

[46] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.

[47] Shao Liu, Tamer Başar, and Ravi Srikant. Tcp-illinois: A loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, pages 55–es, 2006.

[48] Yanwei Liu, Jinxia Liu, Antonios Argyriou, and Song Ci. Mec-assisted panoramic vr video streaming over millimeter wave mobile networks. *IEEE Transactions on Multimedia*, 21(5):1302–1316, 2019.

[49] Josip Lorincz, Zvonimir Klarin, and Julije Ožegović. A comprehensive overview of tcp congestion control in 5g networks: Research challenges and future perspectives. *Sensors*, 21(13):4510, 2021.

[50] M. Luglio, M.Y. Sanadidi, M. Gerla, and J. Stepanek. On-board satellite "split tcp" proxy. *IEEE Journal on Selected Areas in Communications*, 22(2):362–370, 2004.

[51] Imtiaz Mahmud, Geon-Hwan Kim, Tabassum Lubna, and You-Ze Cho. Bbr-acd: Bbr with advanced congestion detection. *Electronics*, 9(1):136, 2020.

[52] I. Maki, G. Hasegawa, M. Murata, and T. Murase. Throughput analysis of tcp proxy mechanism. In *Institute of Electronics, Information and Communication Engineers Technical Report*, pages 07–12, 2004.

[53] Ichinoshin Maki, Go Hasegawa, Masayuki Murata, and Tutomu Murase. Performance analysis and improvement of tcp proxy mechanism in tcp overlay networks. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 1, pages 184–190. IEEE, 2005.

[54] Uttam Mandal, M Farhan Habib, Shuqiang Zhang, Christoph Lange, Andreas Gladisch, and Biswanath Mukherjee. Adopting hybrid cdn–p2p in ip-over-wdm networks: an energy-efficiency perspective. *Journal of Optical Communications and Networking*, 6(3):303–314, 2014.

[55] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 287–297, New York, NY, USA, 2001. Association for Computing Machinery.

[56] Matt Mathis and Jamshid Mahdavi. Deprecating the tcp macroscopic model. *ACM SIGCOMM Computer Communication Review*, 49(5):63–68, 2019.

[57] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[58] Microsoft. Azure stack edge. https://azure.microsoft.com/en-us/products/azure-stack/edge, 2024. Accessed: 2024-08-07.

[59] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate iot edge computing with lightweight virtualization. *IEEE Network*, 32(1):102–111, 2018.

[60] Ning Mu, Shulei Gong, Wanqing Sun, and Quan Gan. The 5g mec applications in smart manufacturing. In *2020 IEEE International Conference on Edge Computing (EDGE)*, pages 45–48, 2020.

[61] Charles Kihungi Njogu, Wang Yang, Humphrey Waita Njogu, and Adrian Bosire. Bbr-with enhanced fairness (bbr-efra): A new enhanced rtt fairness for bbr congestion control algorithm. *Computer Communications*, 200:95–103, 2023.

[62] Sudheer Poojary and Vinod Sharma. An asymptotic approximation for tcp cubic. *Queueing Systems*, 91:171–203, 2019.

[63] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. Edge computing for the internet of things: A case study. *IEEE Internet of Things Journal*, 5(2):1275–1284, 2018.

[64] Netfilter Project. iptables: Administration tool for ipv4/ipv6 packet filtering and nat. https://www.netfilter.org/projects/iptables/. Accessed: 2024-08-1.

[65] M Rajesh and JM Gnanasekar. A systematic review of congestion control in ad hoc network. *International Journal of Engineering Inventions*, 3(11):52–56, 2014.

[66] Jagruti Sahoo, Mohammad A. Salahuddin, Roch Glitho, Halima Elbiaze, and Wessam Ajib. A survey on replica server placement algorithms for content delivery networks. *IEEE Communications Surveys & Tutorials*, 19(2):1002–1026, 2017.

[67] Nicolas Keiji Cattani Sakashita, Maurício Aronne Pillon, Charles Christian Miers, and Guilherme Piêgas Koslovski. Investigating the impact of congestion control algorithms on edge-cloud continuum. In *International Conference on Advanced Information Networking and Applications*, pages 26–37. Springer, 2024.

[68] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. *SIGOPS Oper. Syst. Rev.*, 36(SI):315–327, dec 2003.

[69] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[70] Charalambos Sergiou, Pavlos Antoniou, and Vasos Vassiliou. A comprehensive survey of congestion control protocols in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 16(4):1839–1859, 2014.

[71] Amazon Web Services. Aws outposts family. https://aws.amazon.com/outposts/, 2024. Accessed: 2024-08-07.

[72] Amazon Web Services. Aws wavelength. https://aws.amazon.com/wavelength/, 2024. Accessed: 2024-08-07.

[73] Miguel Landry Foko Sindjoung, Mthulisi Velempini, and Alain Bertrand Bomgni. A mec architecture for a better quality of service in an autonomous vehicular network. *Computer Networks*, 219:109454, 2022.

[74] Ashish Singh, Suresh Chandra Satapathy, Arnab Roy, and Adnan Gutub. Ai-based mobile edge computing for iot: Applications, challenges, and future scope. *Arabian Journal for Science and Engineering*, 47(8):9801–9831, 2022.

[75] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 479–490, New York, NY, USA, 2014. Association for Computing Machinery.

[76] Jihoon Sung, Minseok Kim, Kyongchun Lim, and June-Koo Kevin Rhee. Efficient cache placement strategy for wireless content delivery networks. In *2013 International Conference on ICT Convergence (ICTC)*, pages 238–239. IEEE, 2013.

[77] Igor Sysoev. nginx. https://nginx.org/en/. Accessed 2024-06-10.

[78] Sihai Tang, Bruce Chen, Harold Iwen, Jason Hirsch, Song Fu, Qing Yang, Paparao Palacharla, Nannan Wang, Xi Wang, and Weisong Shi. Vecframe: A vehicular edge computing framework for connected autonomous vehicles. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 68–77, 2021.

[79] HAProxy Technologies. Ahaproxy - the reliable, high performance tcp/http load balancer. https://www.haproxy.org/. Accessed: 2024-08-1.

[80] HAProxy Technologies. Haproxy starter guide - version 2.8. https://docs.haproxy.org/2.8/intro.html, 2024. Accessed: 2024-08-07.

[81] A. Vakali and G. Pallis. Content delivery networks: status and trends. *IEEE Internet Computing*, 7(6):68–74, 2003.

[82] Feng Wang, Jiangchuan Liu, Minghua Chen, and Haiyang Wang. Migration towards cloud-assisted live media streaming. *IEEE/ACM Transactions on networking*, 24(1):272–282, 2014.

[83] Kemeng Wang, Shurui Kong, Xuezheng Chen, and Min Zhao. Edge computing empowered smart healthcare: Monitoring and diagnosis with deep learning methods. *Journal of Grid Computing*, 22(1):30, 2024.

[84] Xiang Xiao, Ming Zhao, and Yusen Zhu. Multi-stage resource-aware congestion control algorithm in edge computing environment. *Energy Reports*, 8:6321–6331, 2022.

[85] Kana Yamanegi, Takayuki Hama, Go Hasegawa, Masayuki Murata, Hideyuki Shimonishi, and Tutomu Murase. Implementation experiments of the tcp proxy mechanism. In *6th Asia-Pacific Symposium on Information and Telecommunication Technologies*, pages 17–22. IEEE, 2005.

[86] Jiannan Yang, Tiantian Qian, Fan Zhang, and Samee U. Khan. Real-time facial expression recognition based on edge computing. *IEEE Access*, 9:76178–76190, 2021.