Leveraging the MEC to Accelerate TCP in 5G Networks

Hala Qadi
University of Waterloo, Canada
hqadi@uwaterloo.ca

Martin Karsten
University of Waterloo, Canada
mkarsten@uwaterloo.ca

Samer Al-Kiswany
University of Waterloo, Canada
alkiswany@uwaterloo.ca

***Abstract*—We explore the viability of using edge computing to accelerate end-to-end communication performance in 5G environments through TCP splitting. We explore different scenarios with varying congestion control protocols, packet drop rates, and TCP buffer sizes. The results show that edge computing, especially through TCP splitting, can significantly improve end-to-end communication performance over an unreliable link layer. TCP splitting over a reliable link layer does not bring any benefit and might in fact reduce throughput, due to the unique characteristics of the 5G scheduling and link layer protocols. Furthermore, over an unreliable link layer, TCP splitting brings a higher benefit for flows larger than 64 KB. These findings provide valuable insight into how edge computing can accelerate TCP communication in different network environments.**

***Index Terms*—Multi-access Edge Computing, 5G, TCP Splitting, TCP Optimization, Heterogeneous Networks, Radio Access Network**

I. Introduction

Multi-access edge computing (MEC) is emerging as a crucial component in 5G networks. In 5G networks, the MEC is embedded in the mobile network infrastructure and is typically located near the radio access network (RAN). This proximity enables ultra-low latency processing of client requests. MEC enables low-latency applications, especially in areas such as smart cities, autonomous vehicles, and the Internet of Things (IoT) [1]. Despite its potential to support emerging applications, currently, no production deployment leverages the MEC.

It is challenging to achieve high TCP throughput over wireless networks, due to the heterogeneity of the overall deployment. The communication path has two distinct segments with widely different characteristics. On the radio network, latency is typically low (1 to 5 ms round trip time), due to the limited size of radio cells. However, packet drop rates remain high. In contrast, core networks, between the radio network and the data center, have higher reliability but significantly higher latency (20 to 150 ms round trip time). A TCP connection from a device to the data center will combine the challenging characteristics of both networks: high drop rate with high round trip time (RTT). The recovery time of most congestion control protocols, especially loss-based protocols, increases with high loss rates and high RTT, significantly reducing throughput.

This paper explores the viability of using MEC to improve end-to-end TCP throughput in 5G environments. It focuses on accelerating TCP communications through the use of TCP splitting.

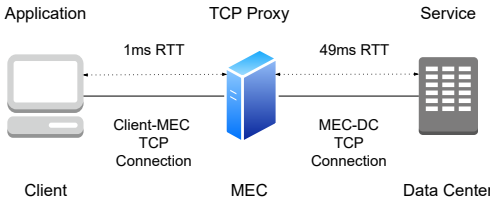


Fig. 1: MEC as an accelerator for TCP over 5G.

TCP splitting (a.k.a splicing) splits a TCP connection into two or more sub-connections. *We posit that using MEC to split a long-distance 5G TCP connection into a sub-connection from the device to the MEC and a sub-connection between the MEC and the cloud data center has the potential to improve TCP throughput.* Figure 1 shows a possible deployment scenario. The data center hosts the service, and the MEC acts as a TCP proxy. When the TCP proxy receives a connection request, it creates a separate TCP connection to the service at the data center. The device-MEC connection has a short RTT and can recover quickly from packet drops, while the long-distance connection from MEC to the data center is more reliable and can achieve higher throughput. Previous research efforts show that TCP splitting improves the communication performance for satellite communications [2].

It is not obvious whether TCP splitting will benefit communication in 5G environments, because 5G uses the hybrid automatic repeat request (HARQ) protocol [3] at the link layer to provide reliable data transmission. Consequently, transport protocols will no see lost packets, but high jitter. Furthermore, 5G uses a special frame structure (Section II-A) the rate limits communication between parties. Moreover, the RTT of the subconnections is different from the ones that have been explored in inter-terrestrial communications [2]; unlike previous studies, the MEC is significantly closer to the user device, in the range of 1-5 ms.

In particular, our aim is to address the following questions. How much throughput improvement can TCP splitting bring under different packet drop rates? Does it perform differently over reliable and unreliable link layers? How do TCP buffer size and congestion control protocol impact throughput? For

what flow sizes does TCP splitting bring benefits? What is the effect of jitter on split TCP connections?

We follow an empirical approach to answer these questions. We use a cluster of three nodes with network emulation to mimic different network conditions. The results show that TCP splitting improves throughput up to 116× over an unreliable link layer. However, TCP splitting over a reliable link layer does not improve throughput. On the contrary, it can lead to up to 58% reduction in throughput in some scenarios (Section IV). This is because 5G scheduling and HARQ cause latency spikes that are higher than the device-MEC RTT. These spikes can lead to a reduction in the congestion window size and communication throughput.

The rest of this paper is organized as follows. Section II provides the necessary background. Section III details the methodology and experimental setup. Section IV presents the evaluation. Section V discusses related work. Finally, conclusions are provided in Section VI.

II. Background

Edge computing is a paradigm in which clusters of computing and storage resources are distributed closer to clients. It has the potential to reduce latency and improve real-time data processing.

The concept of edge computing is as old as the concept of wide-area networking itself. It initially manifested through the deployment of web proxy servers, web caches, and content distribution networks [4]. These scenarios focus on reducing the latency for serving static web content to clients and reducing the stress on a handful of globally distributed data centers. Edge computing for CDN is standard today, with major providers such as Cloudflare [5] and Akamai [6].

A. Edge Computing in 5G

The fifth-generation mobile network (5G) significantly reduces latency and improves throughput. However, with these advances, the tens of milliseconds latency to the nearest data center is unacceptable. 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 lowering service latency. Recent efforts explore using MEC for smart manufacturing, automated guided vehicles, auto visual inspection [7], intelligent video acceleration, video analytics, Internet of Things (IoT) applications [8], and connected vehicles [9].

The fifth generation network [10] utilizes the HARQ protocol at the link layer to provide reliable data transmission alongside a specialized frame structure. In this work, we focus on the Acknowledged Mode (AM) of the link layer, which includes retransmissions, and refer to it as the reliable link layer. The transport layer (layer four of the OSI model) uses well-known transport protocols such as TCP.

5G Frame Structure. 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 [11]. Within each slot, there are resource blocks, which are allocations of frequency and time resources. Devices, normally called 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 it in the next allocated slot. The UE must wait until new resource blocks are allocated, depending on available resources. Hence, the time the UE would need to wait varies [12].

Hybrid Automatic Repeat Request (HARQ). HARQ is an error correction protocol used within the link layer in modern 5G communications [13], [14]. HARQ is designed for communications over channels with high packet error rates, such as wireless networks. In HARQ, packets are encoded using forward error correction (FEC) code. If the receiver experiences a timeout or detects an error that cannot be corrected by FEC, it requests a retransmission of the packet. The sender retransmits the data in the next available slot for that UE. The next available slot can be found in the same radio frame if the network is not heavily utilized, or it can be found in the following frames. Consequently, data loss or corruption manifests as communication jitter to upper transport protocols. The jitter can be small if the packet is retransmitted in the same radio frame, or 10 ms or more if the network is busy and retransmission happens in the following frames. The impact of this fundamental difference on congestion protocol operations has not been studied before.

B. Congestion Control

Congestion control protocols dynamically adjust the sender's transmission rate based on real-time feedback from the network. Congestion control techniques can be classified into four main categories:

- Loss-based congestion control protocols use packet loss as a signal to modify transmission rates. Notable examples include Reno (standard TCP), CUBIC [15], and Westwood [16]. CUBIC modifies the traditional TCP window growth function to a cubic form, making it independent of RTT.
- Delay-based congestion control protocols, such as TCP Low Priority (TCP-LP) [17] and TCP BBR [18], interpret packet delays as congestion indicators.
- Hybrid protocols like TCP-Illinois [19] combine both delay and loss as congestion signals.
- Protocols that use Explicit Congestion Notification (ECN) for early congestion detection. These protocols (such as DATA Center TCP (DCTCP) [20]) require support from routers in the network to signal buffer overflow. In ECN-based protocols, the handling of radio drops depends on their design, classifying them as either loss-based or delay-based.

C. 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 separate connections by means of a TCP proxy, as shown in Figure 1. HAProxy [21] and Nginx [22] are two popular TCP proxy implementations.

TCP splitting typically improves recovery time in case of packet loss. For instance, loss-based congestion control protocols, including standard TCP Reno, react to packet loss as a congestion signal. When a loss occurs, the congestion window (cwnd) is reduced, typically to half of its previous value [23]. 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 under-utilize the network and reduce throughput, especially in high bandwidth and 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 per connection 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 [24].

III. Methodology

We follow an empirical approach to study the viability of using the MEC to accelerate TCP communication. The experiments use three nodes at CloudLab [25] to build a setup similar to the one depicted in Figure 1. Each machine has an Intel(R) Xeon(R) CPU E5-2450 single-socket processor with 16 cores at 2.10GHz, 15 GB RAM, and a 10 Gbps NIC. The nodes are running Ubuntu 20.04. Unless otherwise specified, we use TCP CUBIC [15], the Linux default TCP congestion control. By default, TCP is configured with 2 GB buffers to eliminate any impact that buffer size might have on throughput. We use Traffic Control (TC) [26] to emulate different network conditions and iperf3 [27] to generate load and measure end-to-end throughput. We use HAProxy [21] as a TCP proxy.

The rest of this section details our efforts in emulating the HARQ protocol and 5G scheduling, selecting representative congestion control protocols, and selecting the TCP proxy software.

HARQ and 5G Scheduling Emulation. HARQ is a reliable link layer protocol. When a packet is dropped or corrupted, it is retransmitted, leading to higher jitter. We emulate this behavior because there are no open-source implementations of HARQ or 5G framing and scheduling algorithms. A configurable jitter is added to the communication (Section II-A). Specifically, we apply an additional delay to a percentage of packets to emulate the impact of the packets that are retransmitted and experience longer delays. Since Traffic Control does not offer a straightforward way to achieve this, iptables [28] is used to randomly mark a percentage of the packets. Then we use TC to apply the additional delay to marked packets.

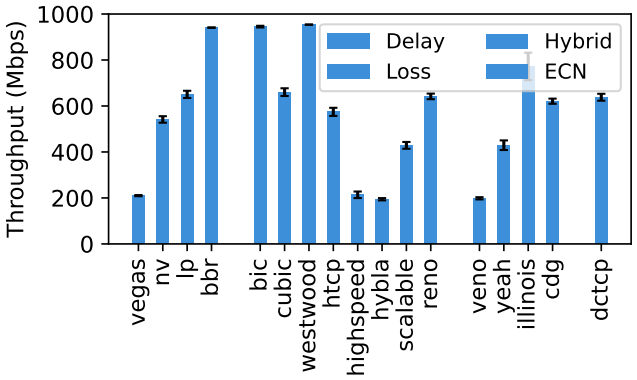


Fig. 2: Average throughput of different congestion control protocols over a 1 Gbps, 0.5% packet drop rate and 1 ms RTT link, the error bars show the standard deviation.

Choice of Congestion Control Protocols. Congestion control protocols fall into four main categories (Section II-B): delay-based, loss-based, hybrid, and ECN-based. Our approach is to select the protocol that achieves the highest throughput as a representative protocol for that category. To this end, we 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.

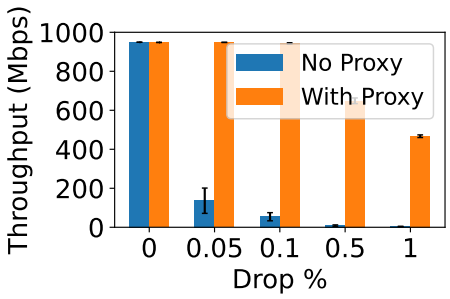
Figure 2 shows the throughput for different congestion protocols at 1 ms RTT. The results show that for delay-based protocols, BBR achieves the highest throughput. The second-best performing protocol is TCP LP. For loss-based protocols, Westwood achieves the highest throughput. Illinois achieves the highest throughput among hybrid congestion protocols. The results with 50ms and 150ms RTT have similar patterns.

Based on the results of this experiment, for delay-based protocols, we select TCP BBR and TCP LP. For loss-based protocols we select TCP Westwood and TCP CUBIC, the latter being the default in Linux systems. For hybrid protocols, we select TCP Illinois. For ECN-based protocols we select DCTCP.

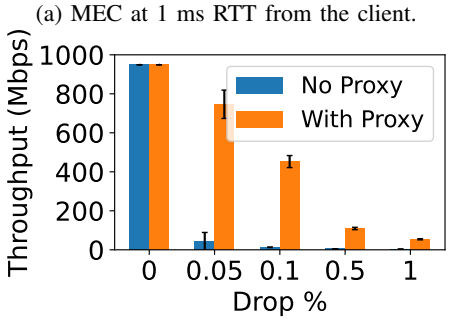
Choice of TCP Proxy. HAProxy is an open-source proxy. HAProxy uses the Linux TCP `splice` system call to forward data between two sockets, thereby avoiding memory copies [29]. We choose to use HAProxy [29] because it directly supports TCP splitting and is the most popular TCP proxy solutions.

IV. Evaluation

We conduct an empirical study to answer the following questions: What is the impact of packet drop on end-to-end throughput over reliable and unreliable link layers? What is the impact of TCP buffer sizes on end-to-end throughput? How do different congestion control protocols perform over both reliable and unreliable link layers? Under what TCP flow



(a) MEC at 1 ms RTT from the client.



(b) MEC at 5 ms RTT from the client

Fig. 3: Average throughput for different drop rates at 50 ms RTT, 1 Gbps over an unreliable link layer. Error bars show the standard deviation.

size does TCP splitting enhance the performance? What is the effect of jitter on split TCP connections?

A. Impact of Packet Drop Over an Unreliable Link Layer

This experiment aims to study the impact of packet drop when the MEC uses TCP splitting over an unreliable link layer between the client and the MEC. The experiment runs on a 50 ms link with a bandwidth of 1 Gbps. A TCP proxy is deployed at 1 ms and 5 ms from the client, and the packet drop rate on the client-MEC link varies across 0%, 0.05%, 0.1%, 0.5%, and 1%.

Figure 3a shows the throughput for different packet drop rates for a proxy placed at 1 ms RTT from the client. 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 a 0.05% drop rate and drops by 99.6% (from 949 Mbps to 4.04 Mbps) with a 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 3b 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 3a and the 5 ms setup in Figure 3b 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 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 client-MEC network has a significant negative impact on TCP communication, and using a TCP proxy significantly improves end-to-end performance.

B. Impact of Packet Drop Over a Reliable Link Layer

We study the impact of packet drop on this deployment over a reliable link layer. The experiment runs on a 1 Gbps link with 50 ms RTT. The jitter is set to 10 ms, based on the assumption that there is a reasonable 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 II-A). The percentage of jittered packets determines how often the client will need to wait the additional 10 ms, which varies between 0%, 0.05%, 0.1%, 0.5%, and 1%. A TCP proxy is deployed at 1 ms and 5 ms from the client. In Section IV-F, we evaluate different jitter values.

Figure 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 a reliable link layer 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 10 ms (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 a 33.7% and 58.1% decrease 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 client-MEC connection, which has an RTT of 1-5 ms. This triggers congestion control mechanisms, resulting in a decrease in the congestion window. Our results with deploying the proxy at 5ms show similar trends.

C. Impact of TCP Buffers over Reliable and Unreliable Link Layers

TCP buffer sizes are known to have a direct impact on TCP throughput, especially in setups with large RTT [30]. This subsection investigates the impact of TCP buffer sizes on the end-to-end throughput over both reliable and unreliable link layers when using TCP splitting. The link is configured with 50 ms RTT and 1 Gbps bandwidth. For the unreliable link layer setup, a packet loss rate of 0.5% is added, while for the reliable link layer, we configure a 10 ms jitter affecting 0.5% of the packets. The MEC running the TCP proxy is deployed at 1 ms RTT from the client.

We experiment with three alternatives:

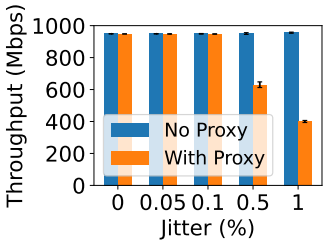
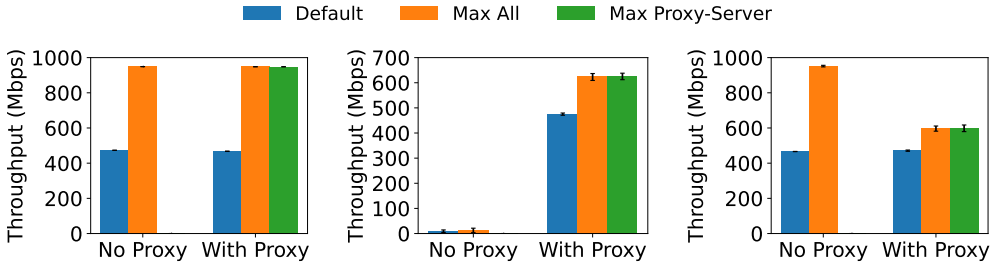


Fig. 4: Average throughput with different jitter percentage.



(a) No packet drop.

(b) Unreliable link layer at 0.5% packet drop rate.

(c) Reliable link layer at jitter affecting 0.5% of packets.

Fig. 5: Average throughput for different buffer configurations at 50 ms RTT, 1 Gbps, and MEC at 1 ms. Error bars show the standard deviation.

- Default: The client, MEC, and data center have default TCP buffer sizes.
- Max Proxy-Server: 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.
- Max All: All nodes are configured with maximum buffer sizes.

Figure 5a shows the average throughput for different buffer configurations without packet drop. When there is no proxy, the default TCP buffer size in Linux under-utilizes the link capabilities, with throughput reaching only 473.2 Mbps, which is 47.32% of the link capacity. The same case with a maximum buffer size increases the throughput, reaching 949 Mbps. With TCP splitting and default sizes, the throughput is 467.6 Mbps, which is comparable to the no proxy with default buffer size. This is because, without losses, the 49 ms RTT proxy-to-server segment is similar to a single 50 ms RTT link. Similarly, increasing the buffer sizes still improves throughput.

Figure 5b shows the average throughput for different buffer configurations over an unreliable link layer with 0.5% packet drop rate. The results show that the network is still under-utilized with the default buffer configuration. Without a proxy, the experiment achieves 9.3 Mbps (0.93% of the link capacity). TCP splitting improves throughput to 475.1 Mbps (47.51% of the link capacity). Although this is a significant improvement compared to the case with no proxy and default buffers, the performance still suffers 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 when using TCP splitting, the size of the buffers on the client side does not impact the end-to-end throughput. This is important since client devices are often memory constrained. The performance of using the maximum buffer size on all machines and only increasing the size on the proxy and server machines is comparable. This is because the client connection has a short 1 ms RTT, hence the default buffers are enough to saturate the link.

Figure 5c shows the average throughput for different buffer configurations over a reliable link layer. 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 a reliable link layer. 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.

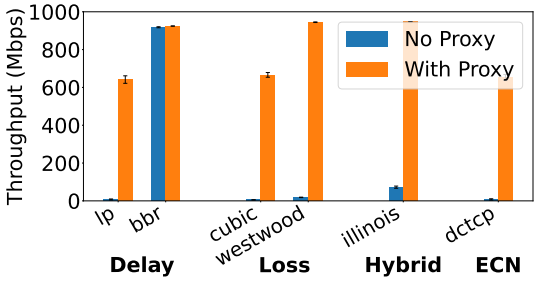


Fig. 6: Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 0.5% drop, and MEC at 1 ms over an unreliable link layer. Error bars show the standard deviation.

We note that throughput with a proxy over an unreliable link layer is comparable to throughput with a TCP proxy over a reliable link layer. With TCP splitting, using maximum buffers increases throughput by 31% over an unreliable link layer, and 27% over a reliable link layer. However, with a reliable link layer and with maximum buffers, throughput with a proxy is still significantly less than that without a proxy. Increasing the buffer sizes does not significantly improve performance, highlighting the substantial impact of packet loss on TCP.

D. Behavior of Congestion Control Protocols Over Reliable and Unreliable Link Layers

We study the behavior of different congestion control protocols. We place a TCP proxy at 1 ms from the client with 0.5% packet loss/jittered.

Figure 6 shows the average throughput for different congestion control protocols over an unreliable link layer. The

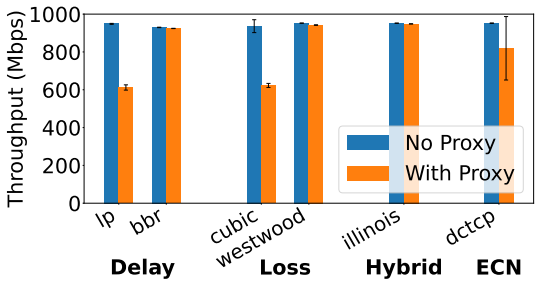


Fig. 7: Average throughput for different congestion control protocols at 50 ms RTT, 1 Gbps, 10 ms jitter affecting 0.5% packets, and MEC at 1 ms over a reliable link layer. Error bars show the standard deviation.

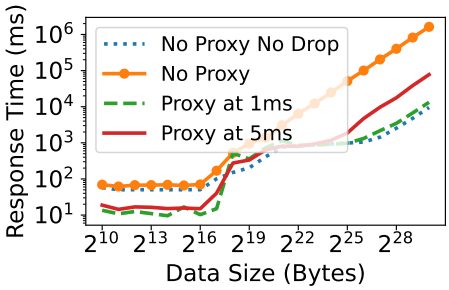


Fig. 8: Average response time for different flow sizes at 50 ms RTT, 1 Gbps, and 0.5% drop over an unreliable link layer

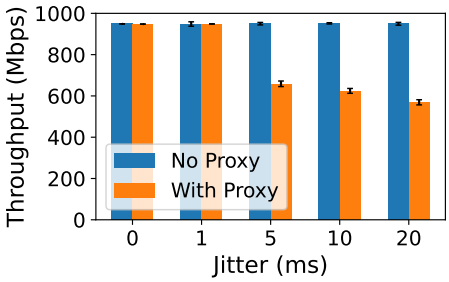


Fig. 9: Average throughput for different jitter values affecting 0.5% of packets at 50 ms RTT, 1 Gbps, and MEC at 1 ms over a reliable link layer. Error bars show the standard deviation.

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, followed by CUBIC and DCTCP, followed by LP.

Figure 7 shows the average throughput for different congestion control protocols over a reliable link layer. TCP splitting results in performance degradation over a reliable link layer. However, with the proxy enabled, Westwood, 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 a reliable link layer.

E. Impact of TCP Flow Size Over an Unreliable Link Layer

TCP splitting significantly improves performance over an unreliable link layer when transmitting large amounts of data. This subsection investigates the flow sizes at which TCP splitting is beneficial.

The experiment reports the time from the moment a TCP socket is opened until the socket is closed successfully, which includes the time to send the data over the network. This measurement is done for different flow sizes. For each size, the experiment runs for at least 60 seconds or until the data is sent at least 10 times.Response time is measured because smaller flow sizes can not fully utilize the network capacity.

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

Figure 8 shows the average response time for different data sizes over an unreliable link layer. For data sizes from 1 KB - 64 KB, the response times in the proxy cases are significantly less 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 that of the with-drop-no-proxy case. Starting from 4 MB, the proxy at 1 ms gets close to the no-proxy-no-drop case. These results show that TCP splitting brings benefit to flows larger than 64 KB over an unreliable link layer.

F. Impact of Jitter Over a Reliable Link Layer

This experiment studies the impact of jitter on throughput over a reliable link layer. In 5G environments, packet drops cause jitter in the communication. The jitter can be small if the packet retransmission happens in the same radio frame or 10 ms or more if it happens in the following radio frames (Section II-A). We experiment on a 1 Gbps link with 50 ms RTT, varying jitter between 0 ms (no dropped packets), 1 ms, 5 ms, 10 ms, and 20 ms to mimic different network loads. The percentage of jittered packets is set to 0.5%. A TCP proxy is deployed at 1 ms from the client.

Figure 9 shows the average throughput for different jitter values. The results show that throughput remains robust without a proxy because packet drops are hidden from TCP, as the increase in RTT (ranging from 0% to 40%) is not sufficient to trigger the TCP congestion control mechanism. TCP splitting is ineffective over a reliable link layer with 0 ms and 1 ms jitter. For the client-to-MEC TCP connection, the increase in RTT (1× for 1 ms jitter) does not trigger congestion control mechanisms. However, for 5 ms, 10 ms, and 20 ms jitter, TCP splitting decreases throughput by 31%, 34%, and 40%, respectively. This is because the additional delay represents 5× to 20× of the RTT on the client-to-MEC (1 ms RTT) connection. In the TCP proxy setup, adding 5 ms jitter achieves only 5% more throughput compared to 10 ms jitter case. This means that there is no significant performance

difference whether the client retransmits the lost packet after 5 ms (within the same radio frame, but after 5 subframes) or after 10 ms in the next radio frame. The results show that the performance degradation depends on the jitter/RTT ratio.

G. Summary

TCP splitting significantly improves end-to-end throughput over an unreliable link layer, especially as packet drop rates increase. Throughput is higher when the proxy is deployed very close to the client (1 ms RTT). However, over a reliable link layer, TCP splitting is ineffective. These observations hold across different congestion control protocols. The reliable link layer 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 $2\times$ to $10\times$ the original RTT, leading to reductions in the congestion window and consequently degrading TCP performance. Performance degradation depends on the jitter/RTT ratio.

TCP Buffer sizes significantly impact performance. The default TCP buffers do not fully utilize the network capabilities. Over an unreliable link layer, increasing 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 a lack of privileged access to modify client configurations.

Over an unreliable link layer, 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 has very low latency.

When the service is deployed directly on the MEC, setups with the unreliable link layer achieve higher throughput at 1 ms RTT. Deployments over a reliable link layer achieve higher throughput at 5 ms RTT.

V. Related Work

A. TCP Splitting

Previous efforts study the impact of TCP splitting on communication performance. Kopparty et al. [31] investigate the impact of splitting long TCP connections to improve fairness. Maki et al. [32], [33] analyze the throughput of TCP overlay networks and highlight that performance degradation becomes quite significant when the network is congested.

There is limited research effort on TCP splitting in 5G networks. Hayes et al. [34] investigate TCP-splitting proxies used in 4G LTE networks and 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. Kim et al. [35] study the performance of TCP splitting in 4G LTE and 3G wireless networks. Their results show that TCP splitting provides average gains of 60% over a 4G LTE network, and 5% over a 3G network. The authors expect greater gains with 5G deployments.

Polese et al. [36] propose milliProxy, a TCP proxy architecture for 5G cellular systems. milliProxy uses cross-layer, proxy-side mechanisms, including flow-window management, MSS optimization, and ACK management, to improve TCP throughput and reduce latency. Lim et al. [37] conduct a broad empirical study of operational 5G networks, examining the effect of edge versus cloud deployment on transport and application performance across multiple transport algorithms. Khorov et al. [38] investiage the performance of TCP and QUIC over 5G and next-generation wireless networks, highlighting the transport challenges caused by highly variable capacity, intermittent blockage, and fast link-quality changes.

Unlike previous efforts, this work is the only study that investigates the impact of the HARQ protocol and the 5G frame structure on 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. TCP splitting proves to be beneficial over an unreliable link layer; however, it is ineffective in 5G networks and can even degrade performance.

B. Theoretical TCP Models

Analytical TCP models [39], [40] were considered instead of using an empirical approach to study the impact of TCP splitting. Unfortunately, there are no accurate models for many congestion protocols used in this study.

Furthermore, existing models [39], [40] 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. In our evaluation, the aforementioned models fail to provide accurate throughput predictions.

VI. Conclusion

The paper explores accelerating TCP communications through TCP splitting in 5G deployments. The results show that edge computing, through TCP splitting, can significantly improve end-to-end communication performance over an unreliable link layer. This is because TCP splitting reduces the impact of packet drop by shortening the feedback loop of the TCP connection. TCP splitting over a reliable link layer does not bring benefits and can reduce throughput. Performance degradation depends on the jitter/RTT ratio. This result contradicts common expectations that TCP splitting would be beneficial in 5G networks [34], [35]. Furthermore, TCP splitting over an unreliable link layer brings higher benefits for flows larger than 64 KB.

These findings provide valuable insights into how edge clusters can accelerate TCP communication in different network environments. Future work could explore optimizing resource usage for TCP splitting. TCP splitting uses edge resources, but it does not bring benefits to all TCP flow sizes. The study shows that TCP splitting is beneficial for certain flow sizes.

VII. Acknowledgment

We thank Ahmed Alquraan for his insightful feedback and technical support. The research team was supported by grants from the National Cybersecurity Consortium (NCC), Natural Sciences and Engineering Research Council of Canada (NSERC) (ALLRP-561423-20 and RGPIN-2025-03332), Ontario Research Fund – Research Excellence program (ORF-RE012-051), Oracle Research Labs, Rogers Communication, and Acronis.

References

[1] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate iot edge computing with lightweight virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[2] M. Luglio, M. Sanadidi, M. Gerla, and J. Stepanek, "On-board satellite "split tcp" proxy," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 2, pp. 362–370, 2004.

[3] R. Comroe and D. Costello, "Arq schemes for data transmission in mobile radio systems," *IEEE Journal on Selected Areas in Communications*, vol. 2, no. 4, pp. 472–481, 1984.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[5] Cloudflare, "Cloudflare," https://www.cloudflare.com/, 2024, accessed: 2024-08-07.

[6] Akamai, "Akamai," https://www.akamai.com/, 2024, accessed: 2024-08-07.

[7] N. Mu, S. Gong, W. Sun, and Q. Gan, "The 5g mec applications in smart manufacturing," in *2020 IEEE International Conference on Edge Computing (EDGE)*, 2020, pp. 45–48.

[8] A. Singh, S. C. Satapathy, A. Roy, and A. Gutub, "Ai-based mobile edge computing for iot: Applications, challenges, and future scope," *Arabian Journal for Science and Engineering*, vol. 47, no. 8, pp. 9801–9831, 2022.

[9] B. Hibat Allah and I. Abdellah, "Mec towards 5g: A survey of concepts, use cases, location tradeoffs," *Transactions on Machine Learning and Artificial Intelligence*, vol. 5, no. 4, 2017.

[10] 3rd Generation Partnership Project (3GPP), "System architecture for the 5g system (5gs)," ETSI, Tech. Rep. TS 23.501 version 18.5.0, 2024. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/18.05.00_60/ts_123501v180500p.pdf

[11] 3rd Generation Partnership Project (3GPP), "5G; NR; Physical channels and modulation," 3GPP, Tech. Rep. TS 38.211 version 18.2.0 Release 18, 2024, accessed: 2024-07-18. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/18.02.00_60/ts_138211v180200p.pdf

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

[13] 3rd Generation Partnership Project (3GPP), "Nr; nr and ng-ran overall description; stage-2," ETSI, Tech. Rep. TS 38.300 version 18.1.0, 2024. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138300_138399/138300/18.01.00_60/ts_138300v180100p.pdf

[14] ETSI, "Nr; medium access control (mac) protocol specification," ETSI, Tech. Rep. TS 38.321 version 17.0.0, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138300_138399/138321/17.00.00_60/ts_138321v170000p.pdf

[15] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[16] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "Tcp westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 287–297. [Online]. Available: https://doi.org/10.1145/381677.381704

[17] A. Kuzmanovic and E. 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)*, vol. 3, 2003, pp. 1691–1701 vol.3.

[18] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[19] S. Liu, T. Başar, and R. 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*, 2006, pp. 55–es.

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

[21] HAProxy Technologies, "HAProxy — the reliable, high performance TCP/HTTP load balancer," https://www.haproxy.org/, 2024, accessed: 2024-08-07.

[22] I. Sysoev, "nginx," https://nginx.org/en/, accessed 2024-06-10.

[23] M. Allman, V. Paxson, and W. R. Stevens, "TCP congestion control," RFC 5681 (Standard), Sep. 2009, accessed: 2024-08-07. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5681

[24] K. Yamanegi, T. Hama, G. Hasegawa, M. Murata, H. Shimonishi, and T. Murase, "Implementation experiments of the tcp proxy mechanism," in *6th Asia-Pacific Symposium on Information and Telecommunication Technologies*. IEEE, 2005, pp. 17–22.

[25] CloudLab, "CloudLab: A cloud testbed for research on the future internet," https://www.cloudlab.us/, accessed: 2024-08-1.

[26] Linux Foundation, "Linux traffic control," https://man7.org/linux/man-pages/man8/tc.8.html, accessed: 2024-08-1.

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

[28] N. Project, "iptables: Administration tool for ipv4/ipv6 packet filtering and nat," https://www.netfilter.org/projects/iptables/, accessed: 2024-08-1.

[29] H. Technologies, "Haproxy starter guide - version 2.8," https://docs.haproxy.org/2.8/intro.html, 2024, accessed: 2024-08-07.

[30] A. Cohen and R. Cohen, "A dynamic approach for efficient tcp buffer allocation," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 303–312, 2002.

[31] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi, "Split tcp for mobile ad hoc networks," in *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, vol. 1. IEEE, 2002, pp. 138–142.

[32] 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*, 2004, pp. 07–12.

[33] ——, "Performance analysis and improvement of tcp proxy mechanism in tcp overlay networks," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 1. IEEE, 2005, pp. 184–190.

[34] D. A. Hayes, D. Ros, and Ö. 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)*. IEEE, 2019, pp. 108–116.

[35] B. H. Kim and D. Calin, "On the split-tcp performance over real 4g lte and 3g wireless networks," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 124–131, 2017.

[36] M. Polese, M. Mezzavilla, M. Zhang, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi, "milliProxy: A TCP proxy architecture for 5G mmWave cellular systems," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 951–957.

[37] H. Lim, J. Lee, J. Lee, S. D. Sathyanarayana, J. Kim, A. Nguyen, K. T. Kim, Y. Im, M. Chiang, D. Grunwald, K. Lee, and S. Ha, "An empirical study of 5g: Effect of edge on transport protocol and application performance," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 3172–3186, 2024.

[38] E. Khorov, A. Krasilov, M. Susloparov, and L. Kong, "Boosting tcp & quic performance in mmwave, terahertz, and lightwave wireless networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2862–2891, 2023.

[39] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[40] S. Poojary and V. Sharma, "An asymptotic approximation for tcp cubic," *Queueing Systems*, vol. 91, pp. 171–203, 2019.