UNIVERSITY
OF OSLO

Master's thesis

Master Essay

TCP Performance Enhancing Proxy to Support
Non-interactive Applications

Joe Bayer

Informatics: Programming and System Architecture
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Autumn 2023

Joe Bayer

Master Essay

TCP Performance Enhancing Proxy to
Support Non-interactive Applications

Supervisors:
Michael Welzl
Kristjon Ciko

Contents

Chapter 1

Intro

Chapter 2

Essay

In this chapter we will present some of the required background knowledge to understand the concepts presented in this paper. Focusing on topics that are outside the common understanding of network programming, especially details of certain congestion controllers and network protocols will be discussed. The rest of the thesis will assume the following topics are known to the reader.

2.1 TCP/IP

Perhaps the most well known internet transport protocol is the Transmission Control Protocol (TCP). It is known for providing reliable and in-order delivery of packets using acknowledgments and re-transmissions [5]. It was first introduced in 1974, but is still the most used internet protocol. However, as the demands of the internet have changed, TCP has not. Though TCP has been updated with minor extensions over the years, such as increased initial windows or new options, the core ideas have stayed the same [2].

Concepts as the end-to-end argument still play a vital role in how TCP is used in the modern internet. TCP is suffering under the illusion that all logic should be placed on the endpoints as the end to end argument denotes. Even if it spans multiple different domains with varying topologies and demands, especially between wired and wireless domains.

- **Wireless Domain**: A wireless communication domain refers to the transmission of data over a wireless medium without the use of physical connections such as wires or cables between devices. This domain covers a variety of technologies, including 3G, 4G, and 5G for mobile communication, Bluetooth and Wi-Fi for close-range communication, and satellite communication for worldwide communication.
- **Wired Domain**: Unlike a wireless domain, a wired domain provides a

steady and reliable bandwidth with low error rates and high throughput. The use of Ethernet and Fiber are typical for wired networks, they enable the transmission of large amount of data over long distances and low signal noise.

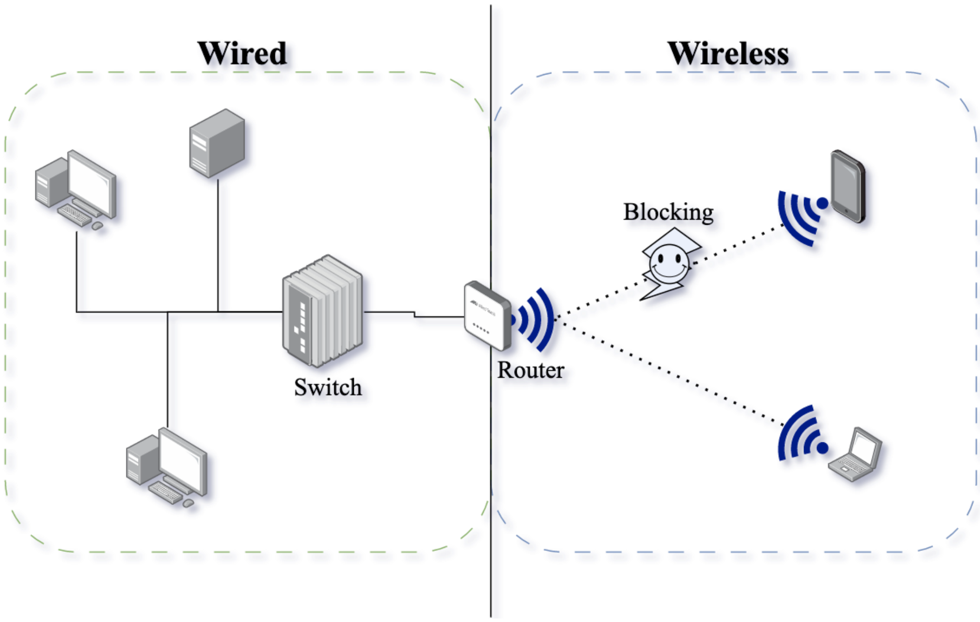

Figure 2.1: Example of network domains

Each domain has different requirements that a single TCP connection cannot provide. Fig. 2.1 shows the two domains and their characteristic differences. Usually, wireless domains experience a lot of changes in connectivity and bandwidths, while the wired domain usually is considered stable. This creates problems for the "modern" TCP which, because of the end to end argument, normally spans multiple domains. Especially congestion control has problems adapting to hight fluctuating bandwidth across long distances and multiple domains.

2.1.1 Congestion Control

Congestion occurs in the internet when a network's resources, such as routers, are overloaded to the point that they diminish quality of the network [11]. Packet loss and high delays are common issues associated to high congestion in the network. To solve the problem of congestion, a distributed algorithm is used: Congestion Control. The main goal of congestion control is to maintain a stable network, while still utilizing the available bandwidth shared among all flows. This is achieved by additively increasing the sending rate, and multiplicatively reducing the sending rate when detecting congestion [10].

Congestion can be detected by packet loss, changes in delay, but also by explicit notifications.

Over time different variations of congestion controller have emerged. Although their goal is the same, reduce congestion in the network, their approaches vary.

- **TCP Reno**: Reno embodies the traditional approach to congestion control. Slowly increasing the sending rate while the network is stable and drastically reducing it on packet loss. TCP Reno was designed for unstable and dynamic networks, where the rapid response rate is crucial to prevent network overloading. However, the slow start rate and aggressive reduction of the sending rate make it sub optimal for more stable networks, where packet loss is less frequent and predictable. Consequently, TCP Reno's reliance on packet loss may lead to unnecessary rate reductions and decreased network throughput.
- **New Vegas**: New Vegas is similar to TCP Reno in most aspects, the main difference is the use of delay to detect congestion instead of packet loss. This makes New Vegas able to react faster to congestion, however it also introduces some interesting side effects. If New Vegas competes with TCP Reno flows, it will start reducing its sender rate before TCP Reno does, this leads to New Vegas losing out on possible bandwidth.
- **Cubic**: Cubic improves on the idea of TCP Reno by using a cubic function to adjust its (congestion window) sending rate in order to achieve higher throughput in a fast manner. Cubic is very efficient in highspeed networks and known for handling large data transfer over long distances. However, Cubic is not as reliable and robust as more traditional congestion controllers like TCP Reno.

In summary, the main differences between TCP Reno, New Vegas and Cubic are their approach to congestion control, their performance in different types of networks, and their trade-off between efficiency and reliability.

2.1.2 3 Way handshake (0 RTT)

For TCP to establish a connection it uses a three-way handshake. Initially, it transmits a synchronization (SYN) packet to the desired endpoint. The endpoint responds with an acknowledgement and a synchronization packet of its own (SYN/ACK). Finally, the client responds with a acknowledgment (ACK). At this point both endpoints have confirmed that they are ready for further communication. For any connection to be established this handshake has to be done. For short flows that terminate in just a few round trips

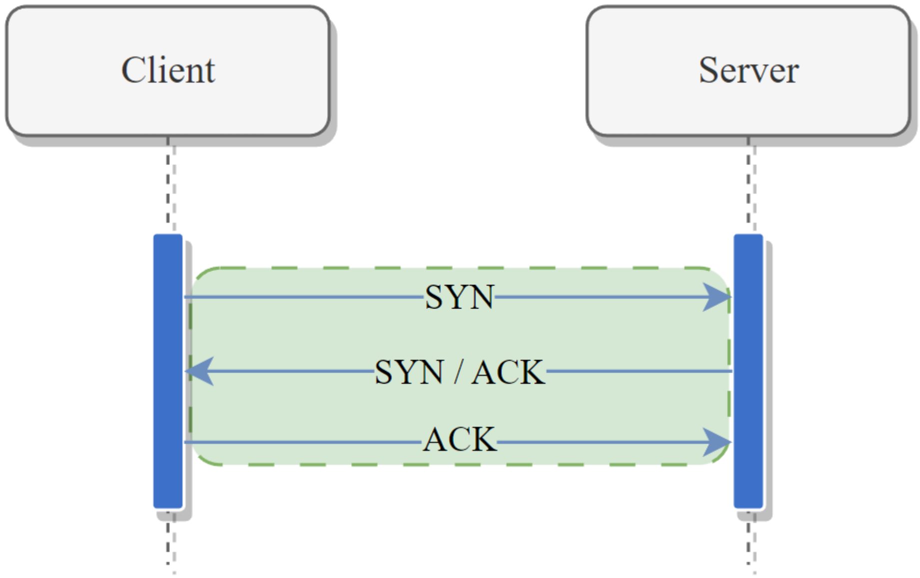

Figure 2.2: The TCP handshake procedure

the initial TCP handshake can be a bottleneck, which is made worse if the connection is using a proxy and has to exchange additional information.

2.1.3 TCP Options and Fast Open

A TCP connection can be configured with optional header extensions called TCP Options [3]. These options change the default behaviour of TCP or add new features. One such feature is TCP Fast Open, which allows data to be added to the initial synchronization packet. A typical use case could be adding a HTTP GET request, thereby saving an entire round trip. In general flows that terminate in a few round trips greatly benefit from this feature. The reason being, the bottleneck in such connections often lies within the initial TCP handshake. Therefore, by removing the extra round trip required to send the first data packet, a significant amount of time can be saved.

TCP Fast Open also has other benefits, such as establishing connections to proxies [2]. When you are trying to establish a connection through a proxy, you get the added delay of a second round trip sending the desired endpoint. This can be avoided by using TCP Fast Open to send the desired endpoint in the first synchronization packet to the proxy. "SYN forwarding" enables the users to establish a proxy connection without any added delays, however it does depend on the user's application to use TCP Fast Open.

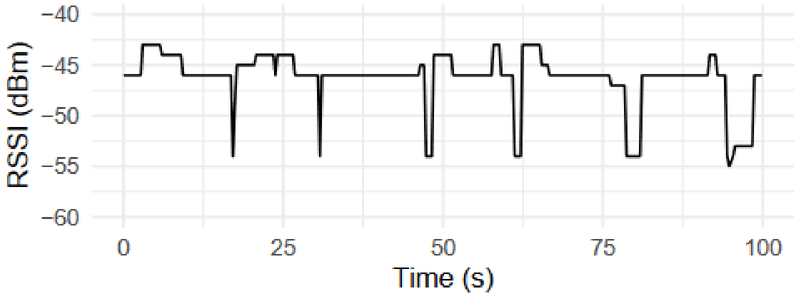

Figure 2.3: 5G bandwidth fluctuations from humans

2.2 Future of wireless communication.

The future of wireless communication has seen a lot of improvements such as highly increased bandwidth achieved through advanced technologies like 5G and beyond. Millimetre frequency bands have opened up new possibilities for wireless communication. These higher frequency bands offer greater capacity and can accommodate more devices, however high frequencies come with a set of new challenges such as highly fluctuating bandwidths. This fluctuation can be influenced by various factors such as signal interference, obstacles in the signal path, and environmental conditions.

2.2.1 5G Millimetre Wave

The emergence of 5G Millimeter wave communications has opened the doors for low latency networks with multiple gigabit bandwidth. This is achieved by using higher millimetre wave (mmWave) frequencies in the range of 30GHz to 300GHz, which as a lot of benefits [1]. A wider spectrum of frequencies to choose from and higher data transfer rates are just some of the many benefits mmWave provides. But along side the benefits, mmWave has also introduced a lot of new challenges.

A big problem with millimetre wave communication is signal path blocking also called "Line of sight blocking" [8]. It's caused by the use of Beam-forming to increase the bandwidth and range of milimeter wave signals. Beam-forming focuses the signal in a certain direction making any blocking of the signal path devastating for the bandwidth. Even the human body can create enough blockage to drastically reduce the bandwidth. This causes huge fluctuations in the bandwidth whenever the signal is blocked.

Fluctuating bandwidths lead to unstable TCP connections with a worst case of losing packets. Current TCP congestion controllers such as CUBIC, New Reno or New Vegas struggle when reacting to sudden fluctuating changes. They are simply not able to utilize the high bandwidth when it is available. Simply increasing the aggressiveness of a congestion controller is not an option either as it would disrupt the internet and not be TCP-friendly. A possible solution could be to buffer packets at the 5G base stations, having the data ready for when the bandwidth is high. This however creates a new problem, bufferbloat.

2.2.2 Buffering

Buffer bloat

The buffer bloat problem occurs when the systems between the endpoints buffer so many packets that the latency drastically increases and the reliability of the network as a whole goes down. The increased latency is detrimental for interactive (latency sensitive) applications. Generally it's preferred to drop packets and keep buffers small to avoid buffering time sensitive packets such as synchronization packets. Although this works in most cases, it's far from a optimal solution.

The increased bandwidth and low latency promises of new technology such as 5G has but a lot of pressure on the efficient forwarding of packets. Small buffers are therefore the standard, But at the same time, fluctuating bandwidth has shown the potential need to buffer packets for non-interactive traffic. Most focus has been on accommodating for latency sensitive applications like virtual reality or remote surgery to name a few.

This thesis will explore non-interactive applications where latency is not that critical and more buffering is acceptable and most likely desirable. By splitting traffic into interactive and non-interactive we can improve the performance of both. By having very small buffers for interactive applications we avoid bufferbloat problems, while utilizing the benefits of big buffers for non-interactive applications.

Packet Scheduling

A method of reducing the effects off bufferbloat is packet scheduling. A system should not send more packets than the weakest link can handle, this idea is built into TCP in the form of congestion control. However, when buffers grow to the point of causing bufferbloat, TCP's congestion control algorithms are unable to confidently determine a sending rate. Packet scheduling can solve this problem as it usually controls the size of the buffers. It makes sure queues can grow when needed, but keep the overall state of the buffers low. Packet scheduling has a lot more to offer than simple queue management, this will be explored later.

Proposed packet scheduling algorithms:

- **FQ CoDel**: The Flow Queue Controlled Delay algorithm, FQ CoDel for short, was partly developed to deal with the bufferbloat problem. Its main goal is to reduce the impact of head-of-line blocking and give a fair share of bandwidth by mixing packets from multiple flows [6]. Internally FQ CoDel uses a FIFO queue, classifying packets into different flows to provide a fair share of bandwidth.

- **HTB**: Hierarchical Token Bucket is a queuing discipline based on assgining different classes a certain amount of bandwidth and sending rate. Because of its extensive bandwidth and delay management it's a good option for testing, espeically in a virtual environment.

2.2.3 Non-Interactive Applications

Non-Interactive applications such as web traffic, file transfers and video streaming can benefit from larger buffering, especially with fluctuating bandwidths. This is because if we are able to buffer the packets closer to their final destination, we have them ready to be sent when the bandwidth changes. By buffering them we can decrease delay times and acheive faster total completation times for non-interactive traffic.(need citation or prove it myself?). At the same time, interactive applications will not suffer under large queue delays that occur under normal buffering.

2.3 Proxy

Proxy servers play a big role in the modern internet, delivering benefits such as anonymity and increased performance [9]. A common use case for a proxy is caching by keeping a copy of popular resources such as a websites. This reduces the latency of accessing the resource as long as the proxy is closer to the user than the original copy. Locality plays an important role in the total latency as any transmission will always be limited by the speed of light.

A proxy can also be used for privacy similar to a Virtual Private Network (VPN). By redirecting network traffic through a proxy, the origin of the traffic appears to be the proxy server rather than the actual end-user. Hypertext Transfer Protocol (HTTP), a popular internet protocol used for accessing websites, has this functionality built in using HTTP tunnels and a special CONNECT method in its header.

```
CONNECT mn.uio.no/:22 HTTP/1.1
```

```
Proxy-Authorization: Basic encoded-credentials
```

2.3.1 PEP

A performance enhancing proxy (PEP) is a connection splitting proxy designed to increase performance of applications using it. The idea behind the PEP is putting more logic such as connection management, buffering, caching inside the network. As the name suggests, a PEP is designed to enhance the performance, but can also introduce new features to a network. An example of a new feature is the multipath support the TCP Transport converter gives. [2].

2.3.2 PEP for wireless communication

Performance enhancing proxies are already deployed and in use for a lot of wireless communication, especially satellites and radio access networks [7]. They have an inherent performance increase just by splitting the connection between the wireless and wired domains. These PEP's are therefore often installed at the base stations. However they are unable to distinguish between interactive or non-interactive traffic, meaning their buffers need to be low and still suffer from fluctuating bandwidth problems.

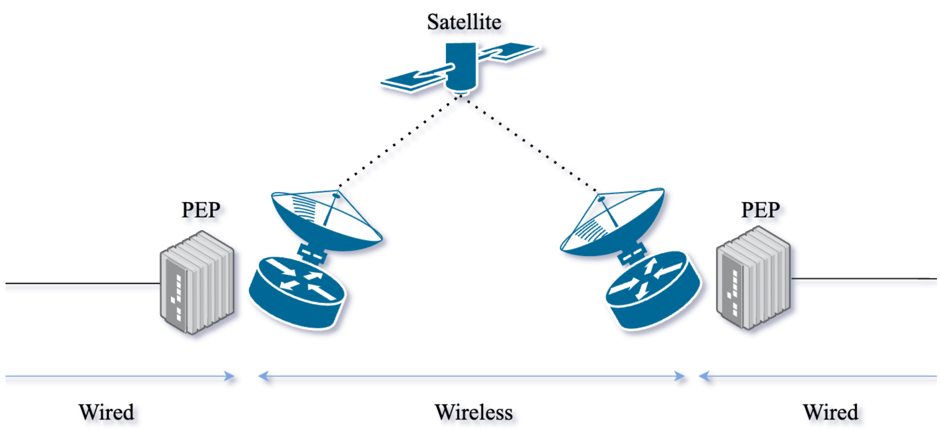

Figure 2.4: PEP installed to support Wireless traffic over satellite.

2.3.3 Transparent vs Non-Transparent

A big discussion regarding PEPs has been if they should be transparent or non-transparent. Transparent PEPs are not visible to the applications that use it. They silently split the connections and spoof the IP-address of both the client and server [4]. This is prone to cause uninteded side effects, such as certain TCP options not being forwarded and security concerns. Non-Transparent PEPs on the other hand are explicitly chosen by either the client or the server, and the sender is aware of the proxy splitting the original connection. This approach can be seen as more ethical and potentially remove some of the stigma associated with PEPs, this however requires modifications at the sender side utilize the PEP.

2.4 Linux

Linux is the most famous open source kernel freely available for anyone to use and modify. Because of the open source nature of Linux, there have been many various operating system implementation based on the Linux kernel. Ubuntu, Fedora or Manjaro are just some of the most famous Linux based operating systems out there. For developers, Linux is the perfect platform to experiment and test their new innovations. You are able to modify and recompile the kernel itself on the fly, and then test the solution on a live operating system. Linux supports most standards and is used by most major corporations such as Facebook, Amazon, Netflix and Google.

2.4.1 Kernel Modules

Thing that makes Linux truly extensible are Loadable Kernel Modules (LKM). Kernel modules are programmes that can be loaded at runtime into the kernel and run with kernel privileges. Running with kernel privileges has a lot of benefits such as having access to internal structures and kernel symbols. Most drivers in the Linux kernel are written as kernel modules as they need access to the system internals.

Congestion controllers and packet schedulers are also usually implemented as kernel modules. That is because Linux exposes a struct with function pointers that can be overwritten by a module. Making the kernel call the new functions instead. Because kernel modules run as part of the kernel they do not need to use system call to do basic I/O as using sockets. Removing the overhead of system calls makes the kernel modules run much faster than default userspace programs.

However, the using Linux kernel modules has the drawback that the program is bound to Linux. The modules will only work in the context of the Linux kernel as they depend on the internal functions, and that they are part of the kernel. Most other operating systems like MacOS will not allow user defined modules to run with kernel privileges. Additionally, any bugs or error in the kernel module with make the entire kernel panic, which usually requires a complete system restart to fix.

Chapter 3

Implementation | Design

```
int pep_tcp_receive(struct socket *sock, u8* buffer, u32 size)
{
  struct msghdr msg = {
    .msg_flags = MSG_DONTWAIT,
  };

  struct kvec vec;
  int rc = 0;

  vec.iov_base = buffer;
  vec.iov_len  = size;

  printk(KERN_INFO "[PEP] kernel_recvmsg: calling recvmsg \n");
pep_tcp_receive_read_again:
  rc = kernel_recvmsg(sock, &msg, &vec, 1, vec.iov_len,
   MSG_DONTWAIT);
  if (rc > 0)
  {
    tlv_print(buffer);
    printk(KERN_INFO "[PEP] kernel_recvmsg: recvmsg returned %d
   \n", rc);
    return rc;
  }

  if(rc == -EAGAIN || rc == -ERESTARTSYS)
  {
    goto pep_tcp_receive_read_again;
  }

  printk(KERN_INFO "[PEP] kernel_recvmsg: recvmsg returned %d\n
   ", rc);
  return rc;
}
```

Listing 3.1: kernel_recvmsg wrapper for receiving for TCP msgs

3.1 TLVs

3.2 PEP

3.3 PEP Connect

Since are PEP is explicitly addressed we need to change how we connect on the client side. The default approach is to use the **connect** function, our goal is to imitate this functions behavior, return code and parameters. By keeping our connect function as similar to the original we reduce the overhead of switching over to it. The only change to the original connect that we need is to specify type of traffic, interactive or non-interactive.

```
int socket, ret;
struct sockaddr_in s_in;
bzero((char *)&s_in, sizeof(s_in));
s_in.sin_family = AF_INET;
s_in.sin_addr.s_addr = inet_addr(IP);
s_in.sin_port = htons(PORT);

socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

ret = connect(server, (struct sockaddr*) &s_in, sizeof(s_in));
ret = pep_connect(server, (struct sockaddr*) &s_in, sizeof(s_in
    ), PEP_INTERACTIVE);
```

Listing 3.2: PEP connection

Table of design decisions based on different PEP implementations compared to ours. 0RTT, Transparent, Using TLV, Special ACKS, connection splitting.

PEP List				
Implementation	0RTT	Connection Splitting	Special ACKs	Transparent
milliProxy	AF	AFG	004	x
PEPDNA	AX	ALA	248	x
SnoopTCP	AL	ALB	008	x
Our PEP	DZ	DZA	012	x
Transport Converter	AS	ASM	016	x
...	AD	AND	020	x
...	AO	AGO	024	x

Chapter 4

Evaluation

4.1 Initial Testbed configuration

4.2 The Spike

4.2.1 Scheduling Algorithms

FIFO s

FQ CoDel s

PFIFO s

4.2.2 Interactive vs PEP

Our first experiment consists of having a interactive flow (100 byte UDP packets at 2kBps) competing with a file transfer, one default end to end and one through our PEP. Highlighting some of the initial differences between an end to end connection and a PEP.

4.2.3 PEP vs E2E tests

The first test consists of evaluating our PEP against a default TCP end to end (E2E) connection, while also highlighting the difference a packet scheduler can make. Fig. **??** shows the results, the red lines represent important events in the timeline. The first represents the start of a file transfer, the second shows a bandwidth change from 10mbit to 75mbit, while the last line shows when the file transfer finished.

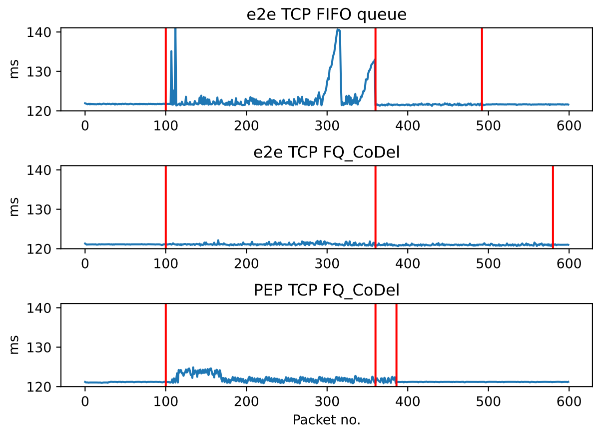

Figure 4.1: Interactive UDP traffic

4.2.4 10x10 tests

To further evaluate the PEP we conducted an experiment where we have 1-10 flows competing end to end versus through our PEP.

4.2.5 Spike

Chapter 5

Conclusion

Bibliography

[1] S. K. Agrawal and Kapil Sharma. 5g millimeter wave (mmwave) communications. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, page 3630–3634, Mar 2016.

[2] Olivier Bonaventure, Mohamed Boucadair, Sri Gundavelli, SungHoon Seo, and Benjamin Hesmans. 0-RTT TCP Convert Protocol. RFC 8803, July 2020.

[3] David Borman. TCP Options and Maximum Segment Size (MSS). RFC 6691, July 2012.

[4] Kristjon Ciko, Michael Welzl, and Peyman Teymoori. Pep-dna: A performance enhancing proxy for deploying network architectures. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–6, 2021.

[5] Wesley Eddy. Transmission control protocol (tcp). Request for Comments RFC 9293, Internet Engineering Task Force, Aug 2022.

[6] Toke Høiland-Jørgensen, Paul McKenney, dave.taht@gmail.com, Jim Gettys, and Eric Dumazet. *The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm.* Number RFC 8290. Jan 2018.

[7] Michele Polese, Marco Mezzavilla, Menglei Zhang, Jing Zhu, Sundeep Rangan, Shivendra Panwar, and Michele Zorzi. milliproxy: A tcp proxy architecture for 5g mmwave cellular systems. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 951–957, 2017.

[8] Cristian García Ruiz, Antonio Pascual-Iserte, and Olga Muñoz. Analysis of blocking in mmwave cellular systems: Application to relay positioning. *IEEE Transactions on Communications*, 69(2):1329–1342, Feb 2021.

[9] W. V. Wathsala, Buddhika Siddhisena, and Ajantha S. Athukorale. Next generation proxy servers. In *2008 10th International Conference*

on Advanced Communication Technology, volume 3, pages 2183–2187, 2008.

[10] Michael Welzl. Network congestion control: Managing internet traffic. *Network Congestion Control: Managing Internet Traffic*, pages 1–263, 05 2006.

[11] Michael Welzl, Dimitri Papadimitriou, Bob Briscoe, Michael Scharf, and Michael Welzl. Open Research Issues in Internet Congestion Control. RFC 6077, February 2011.