TUM SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

eBPF-Assisted Relays for Multimedia Streaming

Daniel Alexander Antonius Pfeifer

TUM SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

eBPF-Assisted Relays for Multimedia Streaming

eBPF-Unterstützung für Multimedia-Streaming-Netzknoten

Author:	Daniel Alexander Antonius Pfeifer
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	Mathis Engelbart, M.Sc.
Submission Date:	15.08.2024

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.08.2024 Daniel Alexander Antonius Pfeifer

Acknowledgments

Abstract

In this thesis we propose a new relay setup for multimedia streaming that allows for avoidance of userspace processing by utilizing BPF programs in the Linux kernel. In a sample implementation we demonstrate the feasibility of this approach by designing a relay that is capable of forwarding packets between a server-side and a client-side QUIC connection while still being able to do adaptive bitrate streaming based on client congestion.
We show that this approach saves processing time and reduces latency compared to userspace processing with the relay still adhering to specifications of the QUIC standard and the 'Media over QUIC' (MoQ) draft. One limitation that is not addressed in depth in this thesis is the need for a de- and encryption hardware offload onto a SmartNIC to allow the BPF program to access the packet payload without any restrictions. Since the QUIC standard is still fairly new we are confident that a solution for a potential hardware offload will be found in future research.

Contents

1 Introduction

Whenever we want to enter today's digital world, one metric is crucial: speed. With very optimized and fast networks in place, we have already reached a point where gaining faster information delivery is highly non-trivial. The need for speed has even led to the development of new standards, such as QUIC (Quick UDP Internet Connections), who aim to improve the shortcomings of the most fundamental protocols of the internet, such as TCP, which have been around more than 40 years.
One other opportunity to cut processing time is closely related to the way computers process data. The ISO/OSI model, which is another foundational concept in networking, provides 'a common basis for the coordination of standards development for the purpose of systems interconnection' [Sta94]. As one can imagine, such a 'common basis', even though conventient for large scale systems, can cause unnecessary overhead. In some cases additional speed-ups can be achieved by using a more application-specific approach. This thesis will consider one of such cases and explore the possibilities of avoiding / delaying certain processing steps in order to incread the overall speed of a network relay.

1.1 Research Question

As already mentioned above the usage of application specific approaches in networking allows for increased packet processing speed. In this thesis we will consider a media streaming scenario that runs on top of QUIC by using the 'Media over QUIC' (MoQ) transport protocol [Int24a]. By using eBPF technology together with kernel hook points provided by the Linux-Kernel, we will explore the possibility of having an eBPF program that handles basic relay capabilities, such as packet forwarding and congestion control. Since the QUIC protocol settles both in kernel and userspace we propose a setup that delays any userspace processing until **after** the packet has been forwarded to the client. This way the raw delay that the packet experiences from the initial media-server to the client is minimized, however, since QUIC is a connection oriented protocol, some additional processing in the eBPF program is needed to ensure that the packets are coherent with the state that the client-side of the QUIC connection expects. Besides designing the setup of a relay that supports such an eBPF speedup,

we will also provide a basic example implementation and evaluate its performance considering packet delay improvement and CPU utilization.

1.2 Scope

TODO

1.3 Structure of this Thesis

TODO

1.4 Source Code Repositories

For the development of the relay and the eBPF program, we have come up with the following repositories:

- **Fast-Relay** [Pfe24b]: This is the main repository providing the eBPF program implementations as well as examples of server, relay and client implementations in Go.
- **Quic-Go Adaptation** [Pfe24a]: This repository is a fork of the QUIC library 'quic-go' [See24] and provides a pain Go implementation of the QUIC protocol. For our thesis we needed to make some adaptations to the library to support some hook points for separate functions specifically designed to handle the underlying eBPF setup with its eBPF-Map usage.
- **MoQ-Transport Adaptation** [PE24]: This repository is a fork of the 'MoQ-Transport' [Int24a] protocol and provides some needed adaptations to our examples. One such adaptation is that the server needs to support a categorization of payloads into different priorities in order for the eBPF program to be able to deliberalty drop packets in case of congestion. Getting these priorities could be as simple as differentiating only between I- and P-frames in a video stream or more complex based on the needs of the application and the granularity of the congestion control.

1.5 Citation Examples

Citation [Int24b]. Citation [Sta94]. Citation [Int24a]. Citation [Rod24]. Citation [W3T24]. Citation [JC20]. Citation [Lie18]. Citation [Yan+20]. Citation [Lan+17]. Citation [PVV21].

Citation [KWF03]. Citation [DB19]. Citation [Tyu22]. Citation [See24]. Citation [Pfe24a]. Citation [Pfe24b]. Citation [PE24]. Citation [Pfe24c].

2 Background and Related Work

2.1 QUIC

The Transmission Control Protocol (TCP) has been used as the backbone of the internet for more than 40 years. It has been designed to be reliable and to provide a connection-oriented way of transmitting data, but the modern environment of the internet with its need for increasing throughput make it hard for TCP to keep up. Limitations in the design and resulting issues like head-of-line blocking have raised demand for a newly designed protocol that can keep up with the modern internet. The 'Quick UDP Internet Connections' protocol, short QUIC protocol, is a transport layer protocol built on top of UDP that is designed to be reliable, cryptographically secure and more performant than TCP. It was intended to be the successor of TCP and it has its origins at Google before the standardization by the IETF began in 2016. QUIC, partly because it operates both in user- and kernel-space, has been designed to allow for a more rapid deployment cycle than TCP. Similar to TCP it is a connection-based protocol that uses TLS for encryption [Lan+17]. Already back in 2018, QUIC was the default protocol for the Google Chrome browser which, at the time, made up 60% of the web browser market [Lie18]. A little over two years later, Facebook, now Meta, was using QUIC for more than 75% of their internet traffic which led to improvements regarding request errors, tail latency and header size [JC20]. As of May 2024, QUIC already made up 8.0% of all internet traffic with support from pretty much every major browser [W3T24; Rod24]. With big players like Google, Meta, Microsoft or YouTube putting emphasis on using QUIC to improve their services, this number is likely to increase even further.

2.1.1 Connections and Streams

Since QUIC is a connection-based protocol, some initial overhead to establish a connection is needed. However the design incorporates some features that aim for a more efficient way of establishing connections, e.g. by using 0-RTT (zero round trip time) handshakes. Latency improvements like the 0-RTT handshake however come at the cost of security, since that opens the door for replay attacks. Another part where QUIC tries to optimize connection management is the use of streams. Streams are designed to be lightweight and can be opened without the need of a handshake. It even goes as

far that a single packet can contain the opening of a new stream, stream data as well as the closing of the stream. This allows for new techniques to improve data transmission and will also be part of the fast-relay setup in this thesis. Aside from streams, apparent since QUIC is based on UDP, it is also possible to send data via unreliable datagrams. This further improves versatility of the protocol and allows for new ways of optimizing data transmission.

2.1.2 quic-go and moqtransport

The implementation of the proposed fast-relay setup will be based on the quic-go library, which provides a pure Go implementation of the QUIC protocol as specified in the standards RFC-9000, RFC-9221 as well as some others which are not that important in this thesis. Together with a modified version of the quic-go library, the fast-relay implementation will also use the moqtransport library. This library brings the 'Media over QUIC' (MoQ) protocol to Go and will be used as a media transport protocol when looking at the impact of fast-relays on adaptive real-time video streaming. The MoQ protocol is being standardized by the IETF since July 2023 and has yet to be finalized.

2.1.3 QUIC and Fast-Relays

The QUIC protocol will be a fundamental part of the fast-relay setup in this thesis, yet the ideas used to make relays faster is not limited to QUIC and can be extended to other protocols as well. QUIC is chosen as an example protocl due to its increasing popularity which offers big potential in early adoption and deployment of fast-relays. Besides that, the existing implementations of QUIC related standards provide a good starting point for an implementation, despite the difficulties that the heavy encryption of QUIC brings with it. To mitigate missing technologies, mainly for offloading QUIC decryption and encryption onto hardware, the existing protocol libraries can also be modified easily to simulate any needed behavior.

2.2 eBPF

In 1992 a technology called 'Berkeley Packet Filter' (BPF) was introduced into the Unix kernel. By using BPF it is possible to attach a small BPF-program to some pre-defined hook points in the network stack of the kernel and filter packets there in a stateless manner. This provided more efficiency since the packets did not need to be copied into userspace anymore but could directly be processed in the kernel. One downside to such an approach however is that BPF-programs are limited by the so-called 'BPF-verifier' which needs to check every BPF-program for safety e.g. to avoid infinite loops or access

to invalid memory from withing kernel space. Today, the initial technology of BPF has evolved into 'extended BPF' (eBPF) and allows for more versatile use cases.

2.2.1 eBPF Hook Points

The Linux kernel offers several hook points where eBPF-programs can be attached to. Two main ones are one to attach eBPF-programs to the Traffic Control (TC) subsystem and another one to attach them to the eXpress Data Path (XDP) subsystem. The XDP hook, which is directly located in the NIC-driver, lies lower in the network stack than the TC-hook, which is located in the link-layer. Despite being higher up in the network stack, the TC-hook has the big advantage that it offers ingress and egress processing while the XDP-hook is available for ingress processing only. This makes the XDP-hook suboptimal for the implementation of fast-relays since they heavily rely on processing packets at egress, after those were redirected from ingress. Figure Figure 2.1 illustrates again the relative positions of the TC and XDP hook points in the network stack.

2.2.2 Traffic Control Queuing Disciplines

The Linux Traffic Control Subsystem uses Queuing Disciplines (qdiscs) to define how packets are handled. TODO

2.2.3 eBPF Verifier

TODO

2.2.4 Important eBPF Concepts

TODO

2.2.5 eBPF and Fast-Relays

TODO

2.3 Adaptive Bitrate Streaming

Multimedia streaming is a big part of the internet and many optimizations have been developed to improve the quality of service for the end-users. This includes considering (in real-time) parts of the clients connection state, such as available bandwith, and adapting the rate at which a server sends data. Such a process is called 'Adaptive Bitrate Streaming' and is employed in many of todays streaming setups. An example setup

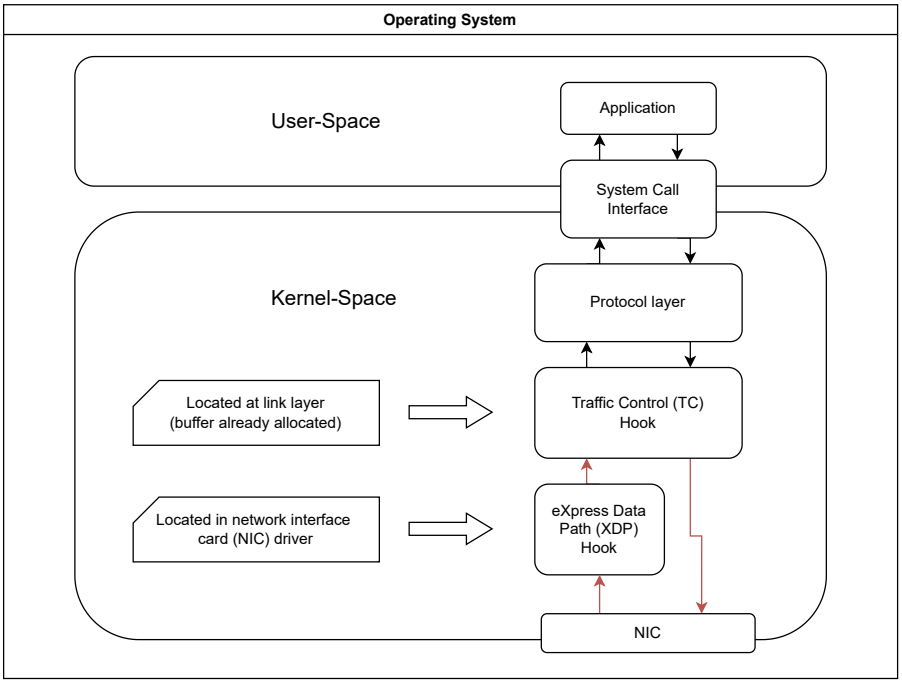


Figure 2.1: Abstracted view of Traffic Control (TC) and eXpress Data Path (XDP) hook points in the Linux kernel network stack. The red loop indicates the 'short-cut' that is utilized by the fast-relay. TC hook allows redirection directly to egress while XDP hook is only available for ingress processing.

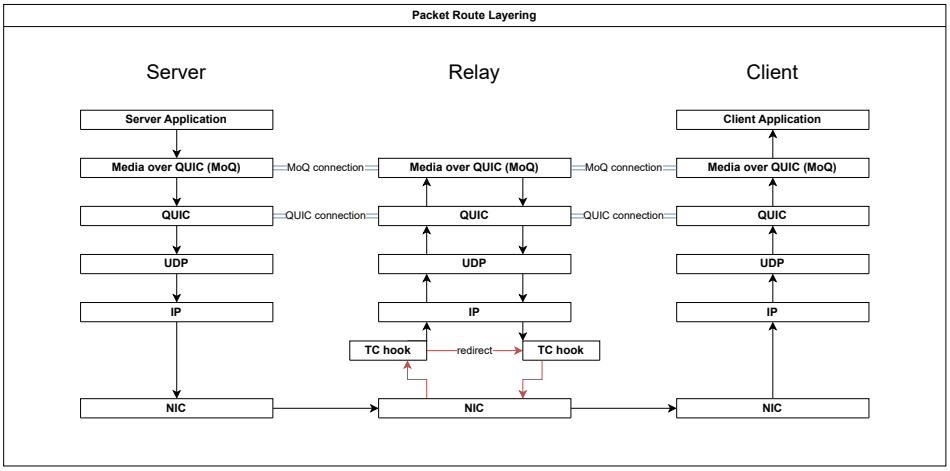


Figure 2.2: Conventional layers of a network stack for client, server and relay. The red loop indicates again the 'short-cut' that is utilized by the fast-relay and based on eBPF packet-forwarding. This avoids the need for the packet to traverse the entire network stack of the relay up to the userspace.

can be seen in Figure 2.3 where within the content delivery network multiple streams with different resolutions exist and the edge server that manages the connection to the client can switch between those streams based on the clients connection state. Youtube and Netflix are examples where, although more complex, similar setups are used to provide a better user experience.

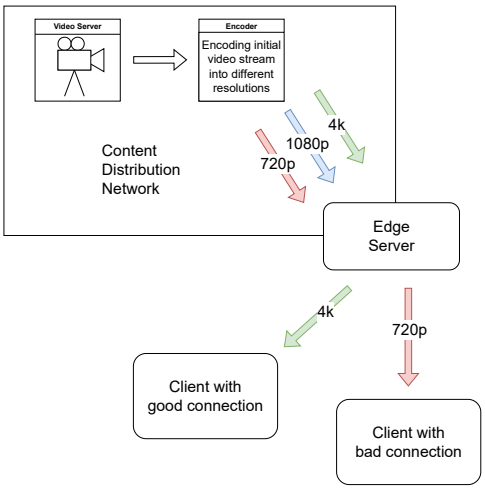


Figure 2.3: A streaming server might send multiple streams with different resolutions to allow adapting to a users bitrate.

The way the example implementation of fast-realys in this thesis is set up, is that the video server will encode into each packet its priority. For example I-frames have a high priority while P-frames have a lower priority. The relay then can decide not to forward certain packets to a client if the client is experiencing congestion.

TODO more specifics on how client state is found out.

2.4 Related Work

2.4.1 eQUIC Gateway

There have been previous publications on making QUIC more efficient by using BPF programs such as [PVV21], where a BPF program is used together with the Linux eXpress Data Path (XDP) to filter packets based on information provided by the userspace. This approach provided significant performance improvements with an increase of throughput by almost a third and a reduction of CPU time consumption

caused by filtering packets by more than 25%. This shows that a setup leveraging Linux kernel features such as BPF has a lot of potential to improve current infrastructure.

2.4.2 Kernel Bypass

Another interesting approach which follows a similar idea of speeding up packet processing by avoiding the Linux network stack is [Tyu22]. The difference in this work is that DPDK is used to bypass the network stack to then process packets in userspace instead of using BPF programs like this thesis does. This, for example, offers more flexibility as the userspace program is not as limited (e.g. by the BPF verifier) as the BPF program but might also lead to slightly more system calls, especially in the setup of a system, when user- and kernel-space need to communicate.

2.4.3 Priority drop

The idea of dropping packets based on their priority to adapt a connection in a congestion event has also been around for a while. [KWF03] explores this in more detail. Mainly improvements like a more tailorable congestion handling than the sole usage of discrete video quality levels as well as an improvement to, potentially randomized, frame dropping are discussed. This thesis, similar to [KWF03], will not focus on how the priority for packets is determined but rather on how those marked packets are handled. For this it is assumed that a higher level protocol has corectly determined the packet priorities and can handle the drop of packets with lower priority in case of limited bandwidth.

3 Fast-Relays

3.1 QUIC Adaptions

TODO

3.2 eBPF Setup

3.2.1 Different BPF Programs

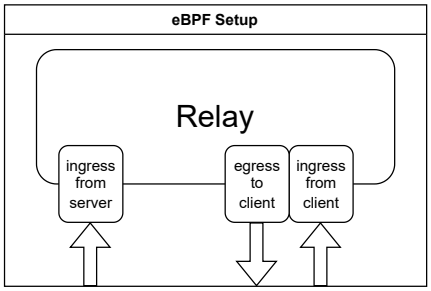


Figure 3.1: The relay has to be equipped with three BPF programs.

In order to allow the relay to forward packets independently of the userspace, we need to equip the relay with three BPF programs as seem in figure. Those three programs are

- a program that handles incoming traffic **from** the clients (client ingress),
- a program that handles outgoing traffic **to** the clients (client egress) and
- a program that handles incoming traffic **from** the video server (server ingress).

Their responsibilities then are

- handling the initial registration of new clients and storing their information such as MAC addresses in a BPF map,
- intercepting the packets from the video server, duplicating and redirecting them to the egress program (as well as sending one unaltered packet to userspace for state management purposes),
- receiving the redirected packets at egress, altering them using the client specific data, deciding (based on packet priority and client congestion) if a packets should be dropped or sent, storing info on sent out packets for future congestion control purposes and finally sending them out to the clients.

This setup allows us to separate any state management and congestion control from the actual packet forwarding and thus makes leaving out any immediate userspace processing possible.

Following is a more detailed description of the responsibilities of each of the three programs.

Client Ingress

TODO

Server Ingress

TODO

Client Egress

The client egress program sees every packet that leaves the relay. This includes packets that have been redirected by the ingress program as well as packets that have been generated by the relay itself. Since the QUIC protocol works with packet numbers for a given connection it is necessary for the egress program to make sure the forwarded packets together with the userspace packets provide a consistent state. For this the egress program maintains its own packet number counter for each connection. That way only one counter has to be maintained and race conditions can be avoided. However, this also means that the packets sent by the userspace are likely to have a different packet number than the one chosen by the QUIC library. This might lead to inconsistencies again but can be avoided by not storing a packet from userspace right away in the packet history but only once the BPF has stored it, along with the changed packet number, in the map used for packet registration. This initially gives a brief window where a packet was sent out but is not saved in the history of the QUIC library but

once the packet is then processed by the userspace routine handling the registration, any incoming ACKs for this packet can be processed correctly. TODO

3.2.2 Packet Registration

In order to make the congestion control algorithm that is running in userspace usable we need to inform the QUIC library about the forwarded packets. This again happens via BPF maps and a separate go routine that continuously polls new entries in the map and processes them. Entries are then added to the packet history to allow the receipt of ACKs. Besides that, the congestion control algorithm will be informed about the forwarded packet in order to be able to react to potential congestion events.

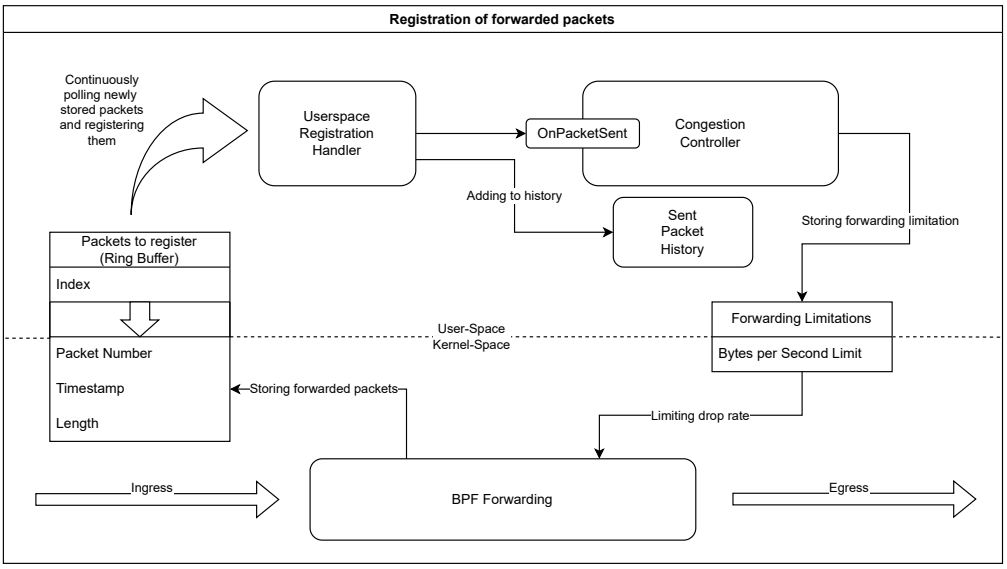


Figure 3.2: Internal setup for registering forwarded packets as well as incorporating forwarding limitations for the BPF program.

TODO: mention

3.3 User Space Avoidance

TODO

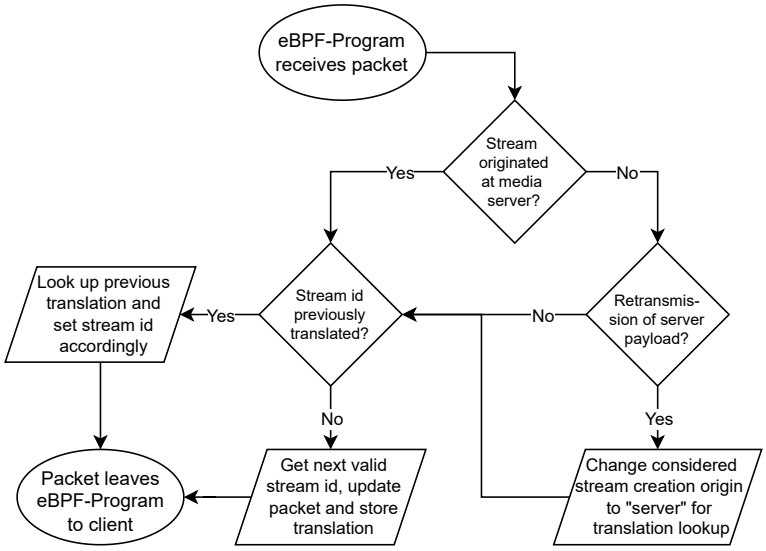


Figure 3.3: TODO (also mention that the check for previous translation is needed since >1 packets per unistream are possible)

3.4 Packet Filtering and Dropping

TODO

3.5 Client Congestion

TODO

3.6 Subscription and State Management

TODO

3.7 Relay Caching

TODO

3.8 Compatibility

TODO

4 Testing

4.1 Setups

TODO

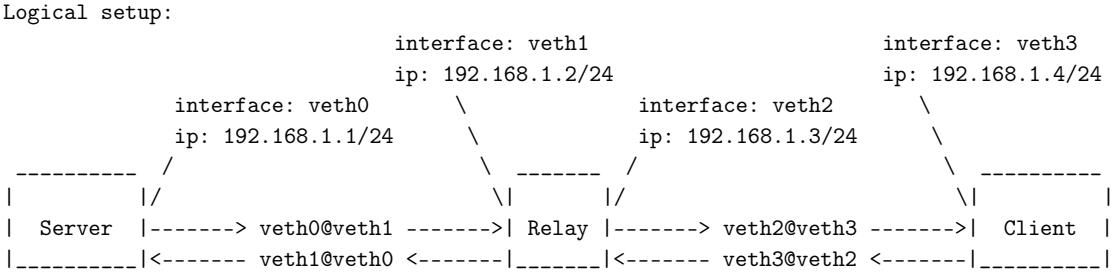
4.1.1 Namespace Environment for Local Testing and Development

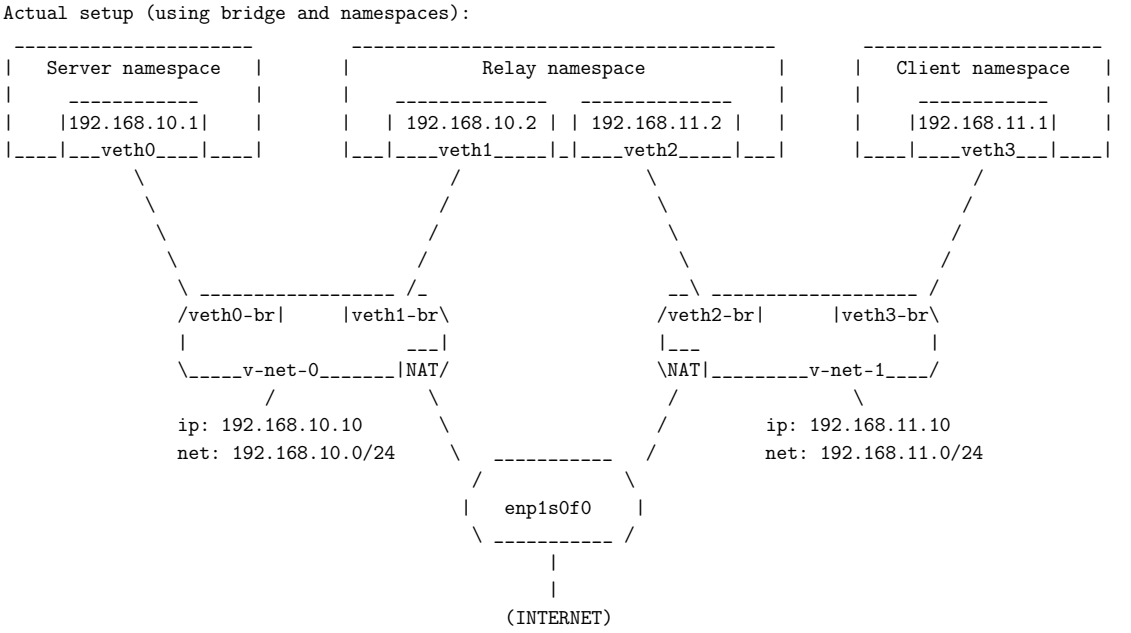
TODO

4.1.2 Physical Server Setup for Real-World Testing

TODO

Logical setup:





4.2 Testing

TODO

4.3 Results

TODO

4.3.1 Delay Reduction of BPF Forwarding

When looking at the impact of eBPF-Forwarding on the delays of packets, we can see that in the namespace environment the delay of a single packet is decreased by around 100 µs when compared to the userspace forwarding. This is shown in Figure 4.1 and consideres the simplest case of the userspace program where it only forwards directly to one connection without the need for much additional computation. Given that the userspace can have arbitrary complex connection management this delay improvement can likely be even bigger in a more sophisticated userspace setup.
Another thing the figure shows is that the delay has a smaller variance due to the fact that the eBPF program path is somewhat similar for each packet whereas, in contrast,

the userspace path can have buffers, queues, or similar that lead to a higher difference in processing time between packets. This effect however might be less observable in a real world scenario due to the ubiquitous network jitter which was influencial in the used namespace environment.

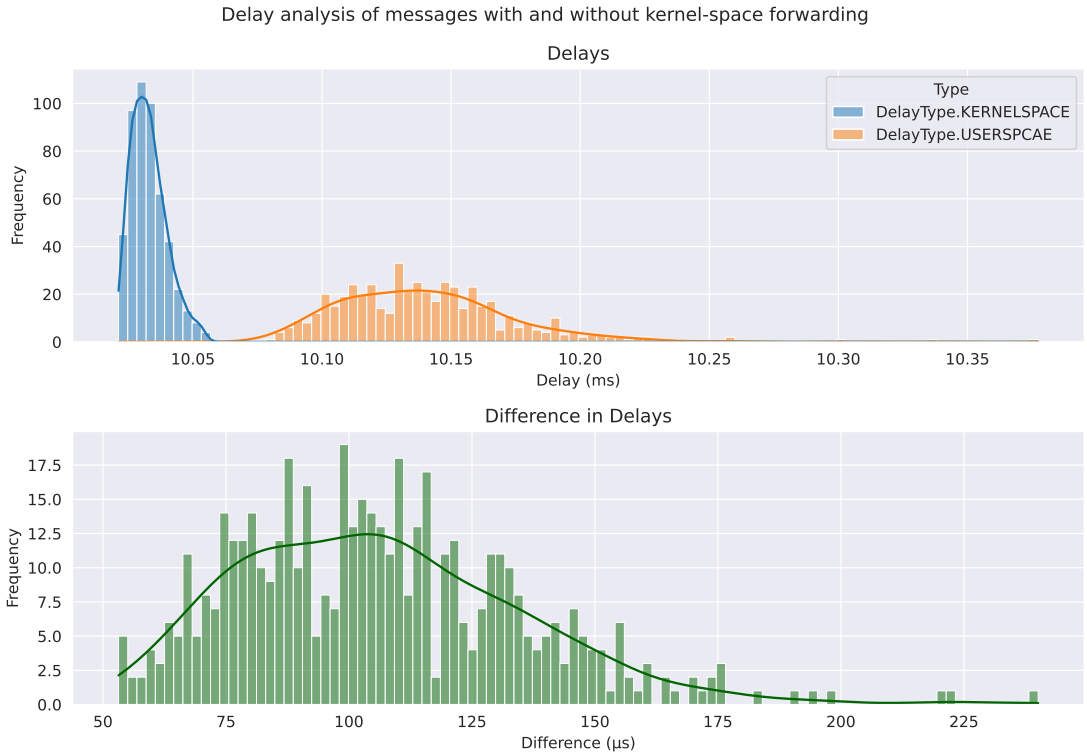


Figure 4.1: By avoiding userspace when processing the 1-RTT packets that contain the payload the delay of a single packet can be reduced. The longer-delay-operations are either handled directly in the eBPF program (e.g. the case for deciding where to redirect a packet to) or handled after the packet has already been sent out (e.g. the case for registering a packet such that the QUIC library knows about it).

5 Future Work

5.1 Hardware Offload

This thesis heavily relies on the fact that the relay can access certain fields (e.g. packet numbers) of the packet, which are generally not accessible without prior decryption. In the current setup, this is made possible by turning off encryption alltogether but to be of any use in a real-world scenario, the encryption of incoming and the decryption of outgoing packets would need to be pushed down below the lowest used BPF hook point in the stack. This means that a hardware offload of encryption and decryption similar to what is done for TCP/IP checksums would be necessary.

Once compatible SmartNIC offload implementations are available one can, besides en- and decryption, also offload the BPF program itself. This then would provide another way of accelerating performance.

Some previous work in this direction has already been done since at least 2019 [Yan+20] but for the purpose of this thesis we did not find any suitable open-source implementation that would allow incorporation into our fast-relay example implementation.

5.2 Compatibility Expansion

This thesis used the QUIC protocol together with media over QUIC (MoQ) to demonstrate how fast-relays that circumvent userspace by utilizing BPF programs can be designed. However, generally speaking the design of fast-relays is not limited to any of these protocols and could be expanded given modifications to necessary fields are possible (i.e. not prevented by encryption) and there is a way to encode the priority of a packet in the packet itself. The latter point could always be realized by using part of the payload which forces a deeper packet inspection within the BPF program but avoids the need to fit the priority into the header of an existing protocol.

6 Conclusion

List of Figures

List of Tables

Bibliography

[DB19] M. Deval and G. Bowers. "Technologies for Accelerated QUIC Packet Processing with Hardware Offloads." Patent EP3541044A1. European Patent application made by Intel Corporation. Sept. 2019.

[Int24a] Internet-Engineering-Task-Force. *Media over QUIC Transport*. Accessed: May 26th, 2024. 2024. URL: https://datatracker.ietf.org/doc/draft-ietf-moq-transport/.

[Int24b] Internet-Engineering-Task-Force. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Accessed: May 26th, 2024. 2024. URL: https://datatracker.ietf.org/doc/html/rfc9000.

[JC20] M. Joras and Y. Chi. *How Facebook is bringing QUIC to billions*. Accessed: May 26th, 2024. Oct. 2020. URL: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[KWF03] C. Krasic, J. Walpole, and W.-c. Feng. *Quality-Adaptive Media Streaming by Priority Drop*. June 2003. DOI: 10.1145/776322.776341.

[Lan+17] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi. "The QUIC Transport Protocol: Design and Internet-Scale Deployment." In: *SIGCOMM '17: Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Aug. 2017), pp. 183–196. DOI: 10.1145/3098822.3098842.

[Lie18] E. Liebetrau. *How Google's QUIC Protocol Impacts Network Security and Reporting*. Accessed: May 26th, 2024. June 2018. URL: https://www.fastvue.co/fastvue/blog/googles-quic-protocols-security-and-reporting-implications/.

[PE24] D. Pfeifer and M. Engelbart. *Media over QUIC Transport*. Accessed: May 26th, 2024. 2024. URL: https://github.com/danielpfeifer02/priority-moqtransport.git.

[Pfe24a] D. Pfeifer. *A QUIC implementation in pure Go*. Accessed: May 26th, 2024. 2024. URL: https://github.com/danielpfeifer02/quic-go-prio-packs.git.

[Pfe24b] D. Pfeifer. *Adaptive Media over QUIC Transport*. Accessed: May 26th, 2024. 2024. URL: https://github.com/danielpfeifer02/Adaptive_MoQ.git.

[Pfe24c] D. Pfeifer. *eBPF-Assisted Relays for Multimedia Streaming*. Accessed: May 26th, 2024. 2024. URL: https://github.com/danielpfeifer02/tum-thesis-bsc-fast-relays.git.

[PVV21] G. Pantuza, M. A. M. Vieira, and L. F. M. Vieira. *eQUIC Gateway: Maximizing QUIC Throughput using a Gateway Service based on eBPF + XDP*. 2021. DOI: 10.1109/ISCC53001.2021.9631262.

[Rod24] L. Rodriguez. *The Future of the Internet is here: QUIC Protocol and HTTP/3*. Accessed: May 26th, 2024. Jan. 2024. URL: https://medium.com/@luisrodri/the-future-of-the-internet-is-here-quic-protocol-and-http-3-d7061adf424f.

[See24] M. Seemann. *A QUIC implementation in pure Go*. Accessed: May 26th, 2024. 2024. URL: https://github.com/quic-go/quic-go.git.

[Sta94] I. O. for Standardization. *ISO/IEC 7498-1:1994*. Accessed: July 3rd, 2024. 1994. URL: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:7498:-1:ed-1:v2:en.

[Tyu22] N. Tyunyayev. *Improving the performance of picoquic by bypassing the Linux Kernel with DPDK*. Master's Thesis. 2022.

[W3T24] W3Techs.com. *Usage statistics of QUIC for websites*. Accessed: May 26th, 2024. May 2024. URL: https://w3techs.com/technologies/details/ce-quic.

[Yan+20] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi. "Making QUIC Quicker With NIC Offload." In: *EPIQ '20: Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Aug. 2020), pp. 21–27. DOI: 10.1145/3405796.3405827.