Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)

Winter Semester 2023

August 11, 2023 – March 03, 2024

Munich, Germany

Editors

Georg Carle, Stephan Günther, Benedikt Jaeger, Leander Seidlitz

Publisher

Chair of Network Architectures and Services

Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)

Winter Semester 2023

Munich, August 11, 2023 – March 03, 2024

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger, Leander Seidlitz

Network Architectures
and Services
NET 2024-04-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Winter Semester 2023

Editors:

Georg Carle
Chair of Network Architectures and Services (I8)
Technical University of Munich
Boltzmannstraße 3, 85748 Garching b. München, Germany
E-mail: carle@net.in.tum.de
Internet: `https://net.in.tum.de/~carle/`

Stephan Günther
Chair of Network Architectures and Services (I8)
E-mail: guenther@net.in.tum.de
Internet: `https://net.in.tum.de/~guenther/`

Benedikt Jaeger
Chair of Network Architectures and Services (I8)
E-mail: jaeger@net.in.tum.de
Internet: `https://net.in.tum.de/~jaeger/`

Leander Seidlitz
Chair of Network Architectures and Services (I8)
E-mail: seidlitz@net.in.tum.de
Internet: `https://net.in.tum.de/~seidlitz/`

Cataloging-in-Publication Data

Seminar IITM WS 23
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, August 11, 2023 – March 03, 2024
ISBN: 978-3-937201-79-5

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2024-04-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2024-04-1
Series Editor: Georg Carle, Technical University of Munich, Germany

Preface

We are pleased to present to you the proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM) during the Winter Semester 2023. Each semester, the seminar takes place in two different ways: once as a block seminar during the semester break and once in the course of the semester. Both seminars share the same contents and differ only in their duration.

In the context of the seminar, each student individually works on a relevant topic in the domain of computer networks, supervised by one or more advisors. Advisors are staff members working at the Chair of Network Architectures and Services at the Technical University of Munich. As part of the seminar, the students write a scientific paper about their topic and afterward present the results to the other course participants. To improve the quality of the papers, we conduct a peer review process in which each paper is reviewed by at least two other seminar participants and the advisors.

Among all participants of each seminar, we award one with the *Best Paper Award*. For this semester, the awards were given to Iheb Ghanmi with the paper *Extracting Information from Machine Learning Models* and Veronika Bauer with the paper *An Overview of the 802.11ax Standard* .

We hope that you appreciate the contributions of these seminars. If you are interested in further information about our work, please visit our homepage `https://net.in.tum.de`.

Munich, April 2023

Georg Carle

Stephan Günther

Benedikt Jaeger

Leander Seidlitz

Seminar Organization

Chair Holder

Georg Carle, Technical University of Munich, Germany

Technical Program Committee

Stephan Günther, Technical University of Munich, Germany
Benedikt Jaeger, Technical University of Munich, Germany
Leander Seidlitz, Technical University of Munich, Germany

Advisors

Jonas Andre (andre@net.in.tum.de)
Technical University of Munich

Max Helm (helm@net.in.tum.de)
Technical University of Munich

Kilian Holzinger (holzinger@net.in.tum.de)
Technical University of Munich

Benedikt Jaeger (jaeger@net.in.tum.de)
Technical University of Munich

Marcel Kempf (kempfm@net.in.tum.de)
Technical University of Munich

Patrick Sattler (sattler@net.in.tum.de)
Technical University of Munich

Johannes Zirngibl (zirngibl@net.in.tum.de)
Technical University of Munich

Leander Seidlitz (seidlitz@net.in.tum.de)
Technical University of Munich

Manuel Simon (simonm@net.in.tum.de)
Technical University of Munich

Markus Sosnowski (sosnowski@net.in.tum.de)
Technical University of Munich

Lion Steger (stegerl@net.in.tum.de)
Technical University of Munich

Florian Wiedner (wiedner@net.in.tum.de)
Technical University of Munich

Lars Wüstrich (wuestrich@net.in.tum.de)
Technical University of Munich

Christoph Schwarzenberg (schwarzenberg@net.in.tum.de)
Technical University of Munich

Seminar Homepage

`https://net.in.tum.de/teaching/ws23/seminars/`

Contents

Block Seminar

Seminar

Containerized Systems: Difference towards network IO

Raphael Auzinger, Florian Wiedner*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: raphael.auzinger@tum.de, {wiedner, andre}@net.in.tum.de

***Abstract*—Containers create an isolated runtime for applications alike and have become an integral part of modern cloud computing. To ensure good isolation and security different runtime architectures emerged. On one side Linux containers like runC and on the other micro VM architectures like Kata Containers, gVisor or Firecracker. These security benefits sometimes come at the cost of additional overhead and resource usage. This work combines and compares various other papers and research which address different architectures and investigate how they influence performance.**

***Index Terms*—containeization, runC, LXD, Kata Container, gVisor, network performance**

1. Introduction

Virtualization and containerization became an important tool in the modern age of developing software and cloud computing. It enables developers to quickly and easily setup systems on their development environment without the necessity to start a virtual machine. Containers can create an environment with all the build tools included for reproducible builds. They are used to run CI/CD tests and verify the project. Further, virtualization enabled cloud service providers to offer not only bare metal products but also parts of a dedicated machine with a variety of configuration options. Starting another virtual private server when the load on a system rises or running edge functions to provide a service, relays on this technology. In a cloud environment speed and efficiency is crucial to reduce latency and energy usage. This might favor one technology over another. Further, containers should be isolated from each other to ensure a secure environment and availability. These two goals are partly in conflict, as we will explore in the following sections.

The paper starts with an introduction to the different standards and how they influence the runtime architectures. In Section 4 the architectures are discussed. How classic containerization with cgroups and namespaces evolved and how optimized micro VMs for enhanced security, performance and seperation from other containers. In Section 5 the networking performance of the different architectures will be compared and analyzed. Section 6 summarizes the paper and will give an outlook on further research and details which can be included in future papers.

2. Related work

This paper is influenced by the research of Wang et al. [1] who analyzed the performance and isolation of runC, gVisor and Kata Containers runtime, Cochak et al. [2], who compared runC and Kata Containers using Docker and on the work of Kovács [3] in which he compared Linux Containers, runC with Docker and KVM virtualization.

3. Containerization Standards

In order to use different runtimes with container images, two standards were established. The Open Container Interface Runtime Specification for the underlying runtime which takes care of execution of a container and the Open Container Interface Image Specification for container images. The OCI Runtime Specification defines the behavior and the configuration interface of low-level container runtimes [4]. Applications like Docker, Kubernetes and Podman [5] [6] [7] can interface with OCI-RS and control the containers. Runtimes which support OCI-RS like runC, LXD, Kata Containers, gVisor and Firecracker can run images which adhere to the OCI Image Spec. It defines the structure of a container image and how the runtime should execute it [8].

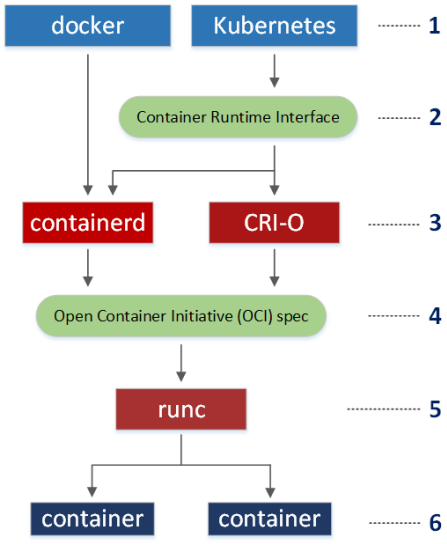


Figure 1: The relationship between Docker, CRI-O, containerd and runc [9]

At level 1. an application like kubernetes communicates with an underlying runtime at level 2. , which

 doi: 10.2313/NET-2024-04-1_01

complies to the Container Runtime Interface Specification. This runtime can now control an OCI image 5. and run it as a new container at level 6. [10].

4. Container Architectures

New architectures were developed around these standards to enhance the security of the host system running containers and to strengthen the isolation between them. Figure 2 shows where different architectures reside, regarding their level of virtualization. On the left no virtualization on a bare metal host system and on the right virtual machines with fully fledged operating systems. Subsection 4.1 4.2 and 4.3 will get into the details and degree of virtualization.

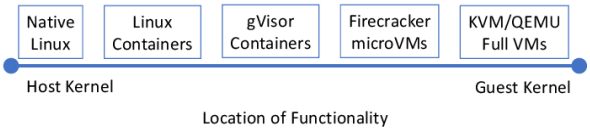


Figure 2: Container isolation spectrum [11]

4.1. Linux Containers

Linux Container, runC and Linux Container Daemon utilize Linux native virtualization methods like cgroups and namespaces to provide a runtime for containers [12] [13] [14]. This approach is very lightweight since the host kernel can be shared with the running container as shown in Figure 3. However, a shared kernel introduces some

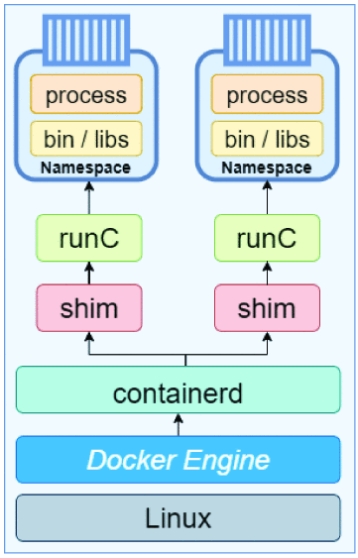


Figure 3: runC virtualization diagram [2]

security concerns if an application escapes the container.

4.2. Sandbox With Multi-purpose Kernel

gVisor tries to solve these security issues and introduces a Sentry and Gofer between the host kernel and a container as shown in Figure 4. System calls inside the container are intercepted and processed by gVisor, which in tern will forward them to the host or abort the call [11].

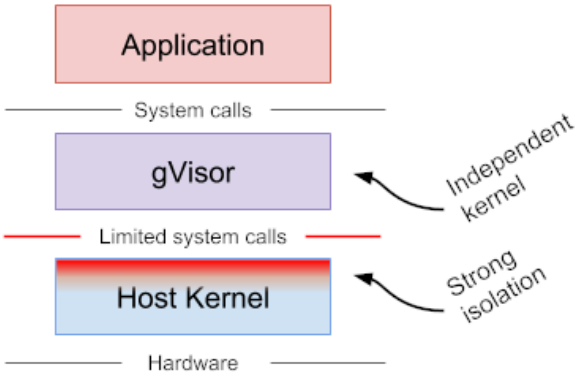


Figure 4: A schematic diagram of gVisor [15]

gVisor handles most of the network tasks inside netstack, but still performs some checks on the host [1]. Netstack is gVisors networking implementation written in Go. All packet processing is done inside gVisors Sentry, which in turn reduces the host I/O syscalls and keeps it isolated from the host networking stack. This virtual networking needs additional processing power. Further the memory management of Go adds additional stress to the underlying system. [15].

4.3. Micro VM Architecture

Kata Containers uses light weight virtual machines to isolate containers, as can be seen in Figure 5. Kernel functionality is moved to a guest operating system or the host QEMU process [11]. These operating systems and kernels are optimized and stripped down to basic functionality to increase start up times and reduce the attack surface [16]. By default, the Kata Containers network uses traffic

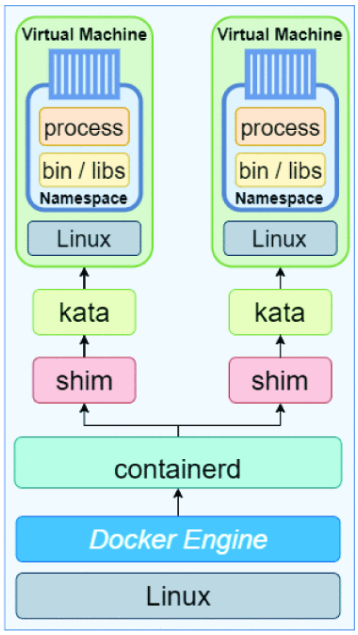


Figure 5: Kata Containers virtualization diagram [2]

control to transparently connect the veth interface with the VM network interface [1].

5. Network Performance

Wang et al. [1] paper looked at the performance between runC, Kata Containers and gVisor. His results are discussed in Subsection 5.1 and are the focus of this paper. Kovács [3] analyzed the networking behavior of Docker compared to LXC and KVM in Subsection 5.2.

5.1. runC - Kata Containers - gVisor

The test was carried out on a machine with an Intel i5-7500 with support for nested virtualization, 8 GB RAM and a 1 TB hard drive. Ubuntu 18.04 LTS (5.4.0) Linux distribution was used as the operating system of choice. On the system were

- Docker version 20.10.1
- KVM version 2.0.0
- gVisor version 20201030.0
- Kata Containers version 1.12.0-rc0

installed [1]. They did not add any restrictions to the containers for the networking tests. Netperf [17] was used to measure network latency and throughput. gVisor was configured to use the user space netstack instead of the host stack [1].

In Figure 6 the TCP_STREAM performance is visualized. runC and Kata Containers are close together and only differ in less than 1%. gVisor has the lowest throughput of the three [1].

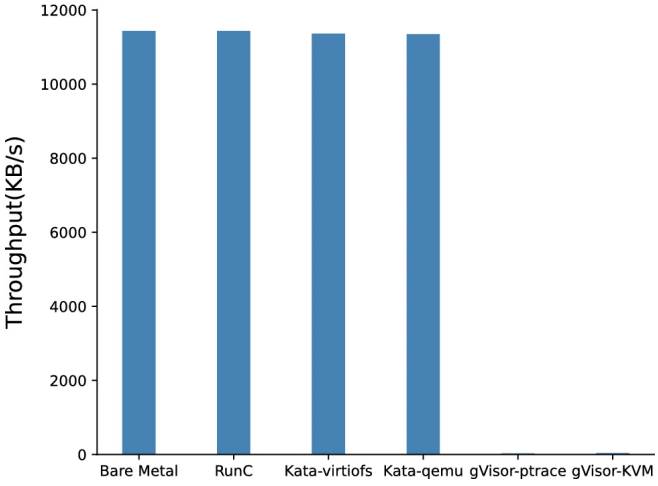


Figure 6: TCP_STREAM network performance [1]

Figure 7 visualizes the TCP_RR, TCP_CRR and UDP_RR performance. TCP_RR are request response tests. The time it takes for a response is measured. While a TCP_CRR test also takes the connection time into account [17]. Bare metal and runC bandwidth performance is almost the same. The loss is less than 1%. Further Kata Containers also looses less than 1% in regard to runC [1] TCP_CRR and UDP_RR show a higher loss of 18.52% and 17.23% respectively. gVisor is again the slowest in terms of network performance. This might be due to its user space networking stack [1].

Figure 8 shows the result of the TCP and HTTP throughput and latency measurements [1]. Kata Containers QEMU TCP and HTTP bandwidth is 1.08% and Kata

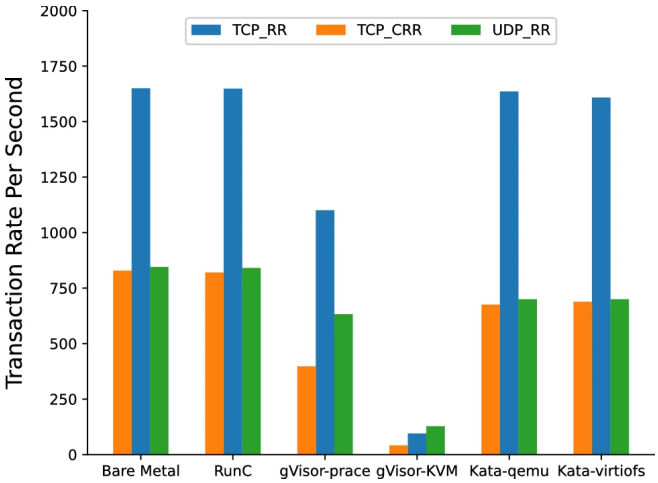


Figure 7: TCP_RR, TCP_CRR and UDP_RR performance [1]

virtiofs 13.12% slower compared to runC. gVisor had the lowest throughput. Under the hood it uses a double packet management mechanism at the Sentry stage (netstack) which leads to the lowest transaction rate and lowest throughput [1].

5.2. Docker - LXD - KVM

The test was carried out on a Huawei CH121 BladeServer with 2x E5-2630 v3 @ 2.40GHz 8Core CPUs and 128 GB DDR4 memory. The server supports nested virtualization. The host operating system was Ubuntu 16.10. The container images ran Ubuntu 16.04.1 LTS. IPerf version 3.1.3 was used to measure network performance. Another HUAWEI BladeServer CH140 with 2x E5-2670 v3 @ 2.30GHz with 12 Cores and 128 GB DDR4 acted as an IPerf server. The two servers were connected via a 10GE HUAWEI CX311 Switch. Docker and LXC were configured with host networking and used the same physical network card as the host operating system [3].

Figure 9 and 10 visualize the results of the measurements. KVM falls behind Linux Container and runC [3]. The throughput is lower and the standard deviation higher due to the higher virtualization effort. It uses a hypervisor and every guest system includes its own guest kernel and virtual networking [18]. This overhead results in a performance penalty. In return, it is more secure since it does not share a kernel with the host OS [18].

6. Conclusion

Each architecture has its drawbacks and benefits. However, each runtime tries to enhance the security of the host system and container isolation. These improvements add layers to the system at the cost of performance.

The research of the cited papers might not reflect the current state regarding networking speed. The overhead might be reduced and speed and security increased, since the projects Kata Containers and gVisor are in active development. runC also got a lot of updates over the past view years, which had an impact on security and networking performance. The results give a rough idea

	Bare metal	RunC	gVisor-ptrace	gVisor-KVM	Kata-QEMU	Kata-virtiofs
TCP Bandwidth (KB/s)	96662.02	96204.8	655.36	686.08	95621.12	86778.88
HTTP Bandwidth(KB/s)	53196.8	49112.58	262.40	268.80	46215.68	43569.15
TCP Latency(us)	593.71	592.29	938.48	900.09	710.96	700.76
HTTP Latency(us)	730.29	776.61	1382.93	1161.59	1015.23	1010.72

Figure 8: TCP and HTTP network performance of container runtimes (Ethr benchmark) [1]

CORRECTED IPERF MEASUREMENT RESULTS

	Docker (MB)	*LXC (MB)*	*Singularity (MB)*	*KVM (MB)*	*Native (MB)*
Average	1122,0	1121,8	1122,1	1116,7	1122,2
Std. Deviation	0,471	0,632	0,316	15,720	0,632

Figure 9: Corrected IPerf Measurement Results [3]

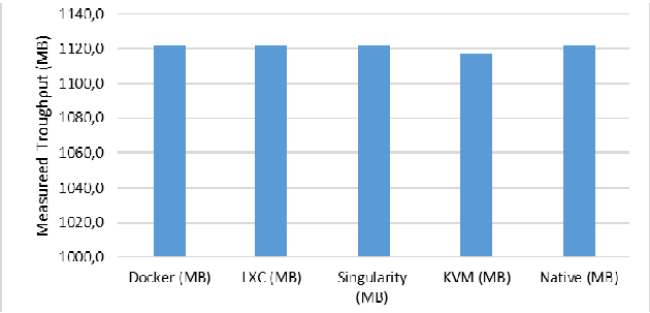


Figure 10: IPerf Corrected Network Measurements [3]

of what can be expected from the different runtimes, but should be taken with a grain of salt.

Different container network drivers could also influence the networking performance gap. Research done by Cochak et al. [2] gives some interesting insights. He compared bridged, host, macvlan and overlay modes with Kata Containers and runC. Some combinations increased the bandwidth but also increase the latency.

Host system optimization was also ignored in this paper. A system tuned for networking / memory / storage performance could lead to higher results or shrink the gap between some runtime approaches.

References

[1] X. Wang, J. Du, and H. Liu, "Performance and isolation analysis of runc, gvisor and kata containers runtimes," *Cluster Computing*, vol. 25, no. 2, pp. 1497–1513, Apr 2022. [Online]. Available: https://doi.org/10.1007/s10586-021-03517-8

[2] H. Z. Cochak, G. P. Koslovski, M. A. Pillon, and C. C. Miers, "Runc and kata runtime using docker: a network perspective comparison," in *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, 2021, pp. 1–6.

[3] Á. Kovács, "Comparison of different linux containers," *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 47–51, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:2864721

[4] opencontainers.org, "Oci runtime spec v1.1," https://opencontainers.org/posts/blog/2023-07-21-oci-runtime-spec-v1-1/, 2023, online; accessed 25-September-2023].

[5] Docker, "Docker docs," https://docs.docker.com/, 2023, online; accessed 25-September-2023].

[6] T. K. Authors, "Kubernetes components," https://kubernetes.io/docs/concepts/overview/components/#node-components, 2023, online; accessed 26-September-2023].

[7] P. team, "What is podman?" https://docs.podman.io/en/latest/, 2023, online; accessed 26-September-2023].

[8] opencontainers.org, "Open containers," https://opencontainers.org/, 2023, online; accessed 25-September-2023].

[9] T. Donohue, "Die unterschiede zwischen docker, containerd, cri-o und runc," https://www.kreyman.de/index.php/others/linux-kubernetes/232-unterschiede-zwischen-docker-containerd-cri-o-und-runc, 2023, online; accessed 25-September-2023].

[10] ——, "The differences between docker, containerd, cri-o and runc," https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/, 2023, online; accessed 25-September-2023].

[11] Anjali, T. Caraza-Harter, and M. M. Swift, "Blending containers and virtual machines: a study of firecracker and gvisor," *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211828470

[12] linuxcontainers.org, "What's lxc," https://linuxcontainers.org/lxc/introduction/, 2023, online; accessed 25-September-2023].

[13] . C. Ltd., "Run system containers with lxd," https://ubuntu.com/lxd, 2023, online; accessed 25-September-2023].

[14] opencontainers.org, "runc readme.md," https://github.com/opencontainers/runc/blob/main/README.md, 2023, online; accessed 25-September-2023].

[15] Google, "What is gvisor?" https://gvisor.dev/docs/, 2023, online; accessed 24-September-2023].

[16] A. W. Services, "Firecracker how it works," https://firecracker-microvm.github.io/#how_it_works, 2023, online; accessed 26-September-2023].

[17] R. Jones, "netperf," https://github.com/HewlettPackard/netperf, 2023, online; accessed 26-September-2023].

[18] C. Ltd., "Kvm hypervisor: a beginners' guide," https://ubuntu.com/blog/kvm-hyphervisor, 2023, online; accessed 26-September-2023].

RFC 9000 and its Siblings: An Overview of QUIC Standards

YaXuan Chen, Benedikt Jaeger*, Johannes Zirngibl*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: chenyaxuan123@gmail.com, jaeger@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—QUIC was a transport protocol designed by Google in 2012, aiming to improve traditional transport protocols, such as TCP. QUIC reduces the handshake latency and solves the head-of-line blocking. The IETF (Internet Engineering Task Force) standardized QUIC in 2021 and continues to improve and extend the QUIC transport protocol. There are 12 RFCs (Request for Comments) related to QUIC, which involve the core concepts of QUIC, QUIC's extensions, applicability, manageability, and HTTP/3. In this paper, we describe the core concept of QUIC based on RFC 9000 and outline four more RFCs to provide an overview of QUIC standards. Furthermore, we discuss which direction the QUIC working group plans for the future.**

***Index Terms*—Transport protocols; QUIC; HTTP/3;**

1. Introduction

With the gradual popularization of the Internet, there has been increasingly more attention on high-speed and stable connections. TCP (Transmission Control Protocol) is the first choice for the developer. However, applications are limited by using TCP as the underlying transport [1]. TCP needs 3 round-trip time (RTT) for setting up the first connection from a client to the server. Furthermore, the head-of-line blocking increases the connection latency when a packet losses [2]. Moreover, TCP is commonly implemented in the operating system kernel. Therefore, the TCP modifications require OS upgrades [3].

QUIC is a new transport protocol designed by Google in 2012 to replace the traditional HTTPS stack: HTTP/2, TLS, and TCP [3], as shown in Figure 1. Notably, QUIC builds on top of UDP and uses a different handshake mechanism from TCP [2], which combines the transport and cryptographic handshake and supports the 0-RTT connection. Furthermore, QUIC solves the head-of-line blocking using a lightweight data-structuring abstraction, streams [3], present in Section 3.1. Additionally, QUIC is implemented in user space, which can be updated without changing the operating system kernel. Applications like Facebook, YouTube, and Gmail have supported QUIC. QUIC is used by 7.6% of all the websites in November 2023 [4]. The proportion of QUIC will continue to increase in the future.

The IETF QUIC Working Group [5] is responsible for the maintenance and evolution of QUIC. The first version of QUIC was standardized in May 2021 as RFC 9000 [6]. There are 12 RFCs related to QUIC until May 2023, as shown in Table 1. However, the content of all RFCs is too much, and we decided to describe part of them in this paper, marked in red in Table 1.

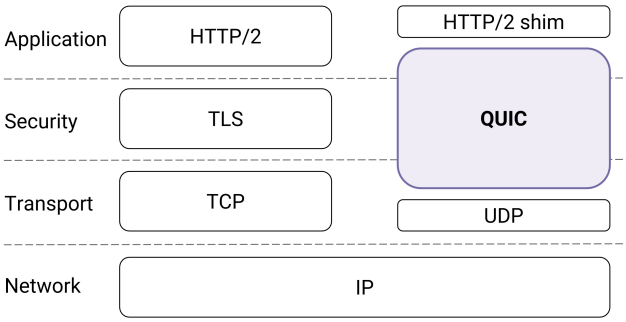


Figure 1: QUIC in the traditional HTTPS stack [3].

The paper is structured as follows: Section 3 describes the core concepts of QUIC. In Section 4, we present an overview of QUIC extensions including RFC 9221 [7], RFC 9368 [8], and RFC 9369 [9]. Furthermore, we present RFC 9114 [10] about HTTP/3. In Section 5, we discuss the future directions of QUIC.

2. Related Work

QUIC related RFCs. This paper provides an overview of QUIC standards and is based on the 12 RFCs [6]–[17].

Core concept of QUIC. Langley et al. in [3] explain the main features of QUIC, and Cui et al. in [1] analyze the advantage of QUIC compared to TCP. Section 3 are closely related to these works.

3. Core Concepts of QUIC

In this section, we outline the core specifications of QUIC from RFC 9000 [6], including QUIC streams, connection migration, and flow control. In addition, we explain how QUIC solves head-of-line blocking. Moreover, we present the congestion control of QUIC from RFC 9002 [7]. Furthermore, we describe the process of 0-RTT handshake from RFC 9001 [12].

3.1. QUIC Streams

QUIC streams are a lightweight abstraction that provides a reliable bidirectional bytestream [3]. Streams consist of multiple stream frames encapsulated in a QUIC packet, as shown in Figure 2. There are two important values in a stream frame:

 doi: 10.2313/NET-2024-04-1_02

TABLE 1: Overview of QUIC-related RFCs

RFCs	Titel	Publication date	Content
8999	Version-Independent Properties of QUIC	May 2021	Definition of the properties of the QUIC transport protocol.
9000	QUIC: A UDP-Based Multiplexed and Secure Transport	May 2021	Description of the core QUIC protocol.
9001	Using TLS to Secure QUIC	May 2021	Functionalities of TLS in QUIC.
9002	QUIC Loss Detection and Congestion Control	May 2021	Loss detection and congestion control mechanisms for QUIC.
9221	An Unreliable Datagram Extension to QUIC	March 2022	QUIC Extension, which supports unreliable datagrams.
9287	Greasing the QUIC Bit	August 2022	Definition of the usage of QUIC Bit.
9368	Compatible Version Negotiation for QUIC	May 2023	The update of the version negotiation mechanisms for QUIC.
9369	QUIC Version 2	May 2023	Definition of the second version of QUIC.
9308	Applicability of the QUIC Transport Protocol	September 2022	Caveats for the developers who want to use QUIC.
9312	Manageability of the QUIC Transport Protocol	September 2022	Guidance for network operations to manage QUIC traffic.
9114	HTTP/3	June 2022	The version of HTTP, which uses QUIC as the transport protocol.
9204	QPACK: Field Compression for HTTP/3	June 2022	The mechanismus of QPACK compression.

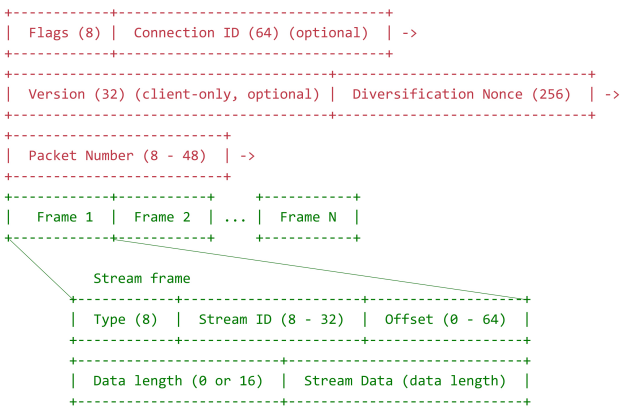


Figure 2: Structure of a QUIC packet [3].

- **Stream ID** is a unique 62-bit integer that identifies the stream. The odd IDs represent client-initiated streams and even IDs for server-initiated streams.
- **Offset** represents in which position the transmitted data should be placed. QUIC packets are encapsulated in UDP packets, and UDP does not transmit data in order. The unordered stream data can be placed in the correct stream and position with stream ID and offset.

Additionally, QUIC also provides unidirectional streams for different purposes (e.g. the control stream in HTTP/3, which is introduced in Section 4.4.)

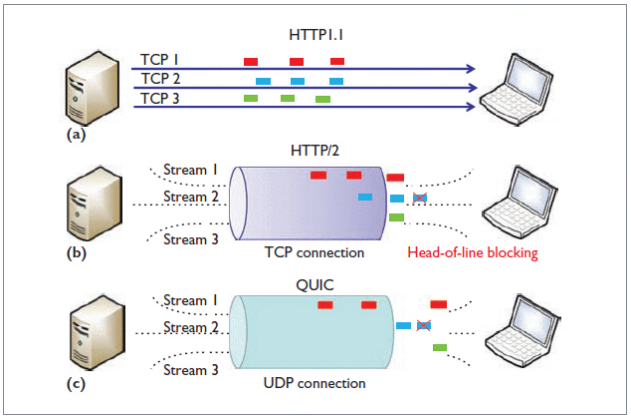


Figure 3: Multiplexing comparison [1].

Stream multiplexing is a method for sending multiple data streams over a single transport connection [1]. Figure 3 provides a comparison of multiplexing:

- **HTTP1.1** creates multiple TCP connections for different resources, as shown in Figure 3a.
- **HTTP/2** improves from HTTP1.1 and multiplexes streams over one TCP connection. However, the data are transmitted in order with TCP, and when one packet is lost on one stream, the packets on other streams are blocked. It leads to head-of-line blocking, as shown in Figure 3b.
- **QUIC** allows other streams to continue to receive packets instead of blocking all streams while waiting for packet recovery [18] because QUIC does not require ordered delivery of all packets, as shown in Figure 3c. Therefore, QUIC eliminates head-of-line blocking.

3.2. Connection Migration

The NAT routers track the TCP connections with the 4-tuple (source IP, source port, destination IP, and destination port). The NAT routers will rebind and change a source IP or port to the client according to the timeout. However, the NAT rebinding is more aggressive for UDP than TCP [3], [19]. Therefore, QUIC introduces a 64-bit Connection ID as an identifier for each connection, which is carried by the QUIC packet, as shown in Figure 2. Endpoints can use Connection ID to track QUIC connections without checking the 4-tuple [20]. At the beginning of the QUIC design, the Connection ID aims to avoid the problem of NAT rebinding. Currently, there is an extension of QUIC in progress, which supports generating routable QUIC Connection IDs [21].

3.3. Flow Control

QUIC provides two levels of flow control:

- **The stream-level flow control** limits the data sent on each stream. The initial limits of the streams are set during the handshake. A receiver can increase the limit periodically to accept more data on the stream by sending MAX_STREAM_DATA frames.
- **The connection-level flow control** defines the total bytes of stream data on all streams, and the limit of connection is determined according to the sum of bytes consumed on all streams [6].

3.4. Congestion Control

Similar to the TCP, QUIC chooses a pluggable congestion control interface, which at first is Cubic [22], to find the suitable algorithms. Notably, QUIC provides a different environment for congestion control than TCP [1]. To improve the design of TCP's sequence number, the QUIC packet is identified by the unique packet number [3]. The receiver reassembles the packet according to the stream offset. This separation of transmission order from delivery order avoids retransmission ambiguities. Moreover, QUIC provides more accurate network roundtrip time estimation with the delay information between when a packet is received and when the corresponding acknowledgment is sent [7]. Furthermore, QUIC supports up to 256 ACK blocks, making QUIC more resilient to reordering and loss than TCP [3]. Consequently, based on the same congestion control, QUIC detects and recovers from loss more efficiently [18].

3.5. 0-RTT Handshake

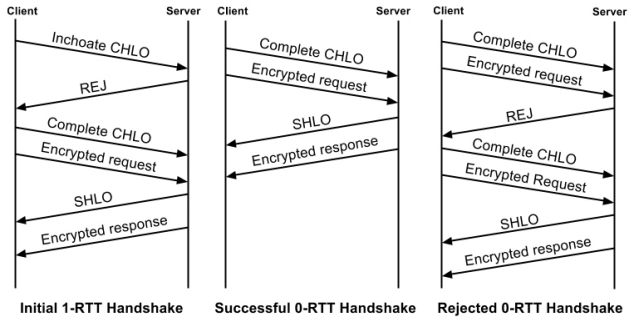


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake [3].

The Transport Layer Security (TLS) protocol provides a secure channel between two communicating peers. Compared to the previous version, TLS 1.3 adds a 0-RTT mode, saving a round trip at connection setup [23]. TCP that uses TLS 1.3 establishes a connection with at least 1-RTT. QUIC combines the cryptographic (TLS 1.3) and transport handshake to minimize the connection establishment time to 0-RTT. There are two types of handshake, as shown in Figure 4:

- **Initial 1-RTT handshake:** For the first connection to the server, the client sends the inchoate client hello (CHLO) message to the server. The server sends the reject message back, which contains valuable information for the subsequent connections [3].
- **Successful 0-RTT handshake:** For the client and server that had communicated before, the client can send complete CHLO with the encrypted request immediately [1].

4. QUIC RFCs

In this section, we discuss QUIC extensions, including RFC 9221 [13], RFC 9368 [8], and RFC 9369 [9]. Furthermore, we introduce RFC 9114 [10], which standardizes HTTP/3, a new version of HTTP that uses QUIC as the transport protocol.

4.1. RFC 9221

RFC 9221 [13] aims to extend QUIC to support the transmission of unreliable datagrams. The demand for transmitting unreliable data is increasing in some areas, such as audio/video streaming, gaming, and real-time network applications. QUIC is the better choice instead of UDP with the following advantages:

- QUIC provides a more precise loss recovery mechanism and more effective single congestion control.
- The application can use both a reliable stream and an unreliable flow within one connection, which benefits from the reduced handshake latency.
- QUIC provides a single congestion control for reliable and unreliable data.

Before RFC 9221 was published, Palmer et al. presented a simple extension to QUIC: ClipStream, used for video streaming [24]. This extension is designed based on a simple observation: not all frames in a video encoding scheme are equally important. ClipStream uses reliable transport for important frames (e.g. I-Frames) and unreliable transport for other frames (e.g. B- and P-Frames). Notably, ClipStream outperforms TCP and QUIC that not support the transmission of unreliable datagrams [24].

4.2. RFC 9368

Initially, QUIC does not provide a complete version negotiation mechanisms. The server can only reject the request from the client that uses an unacceptable version. RFC 9368 [8] updates this mechanism and defines the two types of version negotiation: compatible and incompatible version negotiation

Compatible version negotiation starts when the server knows how to parse the client's first packet, which contains the list of versions that the client knows its first packet is compatible with. The server must select one of these versions it supports as the negotiated version. Subsequently, the server converts the client's first packet into the negotiated version and replies to the client. Notably, version compatibility is not symmetric: A is compatible with B, but B may not be compatible with A.

Incompatible version negotiation happens if the server can not parse the client's first packet. The server sends a version negotiation packet to the client, which contains the server's offered versions. If the client does not find a version that it supports, the connection will be aborted. Incompatible version negotiation causes one more round trip than compatible version negotiation.

4.3. RFC 9369

RFC 9369 [9] defines QUIC version 2 to mitigate ossification concerns and exercise the version negotiation mechanisms, which are presented in Section 4.2. QUIC version 2 provides an example of the minimum changes necessary to specify a new QUIC version. The differences with QUIC version 1 are as follows:

- **Version Field** of long headers is 0x6b3343cf, generated by taking the first four bytes of the sha256sum of "QUICv2 version number".
- **Long header packet types** are different.
- **Cryptography changes** including the salt used to derive initial keys, HMAC-based key derivation function (HKDF) labels, and retry integrity tag.

For more details of the changes, we refer the readers to [9]. QUIC version 2 is not intended to replace version 1 and is compatible with version 1. Notably, a session ticket or token from a QUIC version 1 connection must not used to initiate a QUIC version 2 connection, and vice versa. Moreover, QUIC version 2 provides no different application functionalities than version 1. Furthermore, QUIC version 2 has no changes to the security.

4.4. RFC 9114

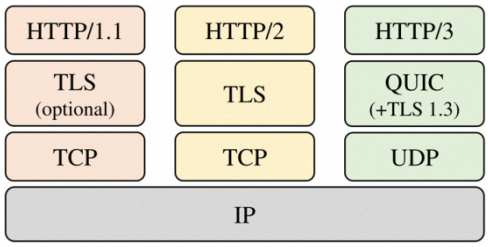


Figure 5: Protocol stack for different HTTP versions [25].

The Hypertext Transfer Protocol (HTTP), born in the early 90s, takes a dominant position in web protocols. Unlike the previous versions, HTTP/3 uses QUIC instead of TCP as the transport protocol, as shown in Figure 5. RFC 9114 [10] defines a mapping of HTTP semantics over QUIC. In this section, we describe how QUIC's features are mapped to HTTP/3 including the discovery of an HTTP/3 endpoint, QUIC streams, and per-stream flow control.

Discovery of an HTTP/3 endpoint: The server can announce the client with the Alt-Svc HTTP response header or the HTTP/2 ALTSVC frame using the "h3" ALPN token, which represents the availability of HTTP/3. The server may serve HTTP/3 on any UDP port. After receiving the Alt-Svc record, the client may establish a QUIC connection using HTTP/3.

QUIC streams: The stream data containing HTTP frame are carried by QUIC stream frames (Section 3.1). QUIC provides bidirectional and unidirectional streams. **Bidirectional streams** are only initiated by the client. Each bidirectional stream handles a pair of HTTP requests and responses. **Unidirectional streams** are used for a range of purposes. Two unidirectional stream types are defined:

- **Control streams** aim to manage other streams. At the beginning of the connection, each side of the endpoint must initiate a single control stream. A pair of single bidirectional streams ensures the performance and stability of data because it allows endpoints to send data as soon as possible.
- **Push stream** is designed for the server push feature introduced in HTTP/2. Server push allows the server to send resources that the client may need to the client without the client's request, as shown by Stenberg in [26].

Flow control: QUIC provides flexible flow control. The two levels of flow control, which are mentioned in Section 3.3, allow the developer to implement several mechanisms. The developer can send the MAX_STREAM_DATA frames in a linear, non-linear, or dynamic manner based on RTT estimates and application data consumption rate, as shown by Marx et al. in [27]. The mechanism of the QUIC flow control is still an open problem and can be improved.

5. Conclusion and Future Work

In this paper, we provide an overview of QUIC and its related RFCs. QUIC solves the head-of-line blocking. Furthermore, the connection can survive during the change of IP and port. Moreover, with the integration of TLS 1.3, the security is improved, and 0-RTT is facilitated to gain a faster handshake. Notably, QUIC shows excellent potential in extensibility. For unreliable connections, QUIC provides more choices for some areas, such as gaming and video streaming. With the promotion of the compatible version negotiation, version 2 of the QUIC is standardized, which provides an example of the new QUIC version. In Addition, QUIC has desirable features in transport for HTTP [10].

The IETF QUIC Working Group continues to contribute to the QUIC, and there are five directions:

- **The qlog**, an extensible high-level schema for a standardized logging format, is now used for the QUIC, HTTP/3, and QPACK events.
- **Connection migration** is going to be improved. The routable QUIC connection IDs are designed to route the packets with migrated addresses correctly.
- **Multipath extension** aims to enhance the usage of multiple paths for a single connection.
- **Control of delaying of acknowledgments** aims to improve connection and endpoint performance.
- **Reliable QUIC stream resets** allow sending the stream data up to a certain byte offset after resetting a stream.

We refer readers for more details to the in-progress documents [5], authored by the IETF QUIC Working Group.

References

[1] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with quic: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.

[2] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?" in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[3] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.

[4] [Online]. Available: https://w3techs.com/technologies/details/ce-quic

[5] [Online]. Available: https://quicwg.org/

[6] J. Iyengar and M. Thomson, "RFC 9000 QUIC: A UDP-based multiplexed and secure transport," 2021.

[7] J. Iyengar and I. Swett, "RFC 9002: QUIC Loss Detection and Congestion Control," 2021.

[8] D. Schinazi and E. Rescorla, "RFC 9368: Compatible Version Negotiation for QUIC," 2023.

[9] M. Duke, "RFC 9369: QUIC Version 2," 2023.

[10] M. Bishop, "RFC 9114: HTTP/3," 2022.

[11] M. Thomson, "RFC 8999: Version-Independent Properties of QUIC," 2021.

[12] M. Thomson and S. Turner, "RFC 9001: Using TLS to secure QUIC," 2021.

[13] T. Pauly, E. Kinnear, and D. Schinazi, "RFC 9221: An Unreliable Datagram Extension to QUIC," 2022.

[14] M. Thomson, "RFC 9287: Greasing the QUIC Bit," 2022.

[15] M. Kühlewind and B. Trammell, "RFC 9308 Applicability of the QUIC Transport Protocol," 2022.

[16] ——, "RFC 9312 Manageability of the QUIC Transport Protocol," 2022.

[17] C. Krasic and M. Bishop, "RFC 9204: QPACK: Field Compression for HTTP/3," 2022.

[18] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols," pp. 290–303, 2017.

[19] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 260–266.

[20] [Online]. Available: https://blog.cloudflare.com/the-road-to-quic/

[21] [Online]. Available: https://quicwg.org/load-balancers/draft-ietf-quic-load-balancers.html

[22] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for fast long-distance networks," 2018.

[23] E. Rescorla, "Rfc 8446: The transport layer security (tls) protocol version 1.3," 2018.

[24] M. Palmer, T. Krüger, B. Chandrasekaran, and A. Feldmann, "The quic fix for optimal video streaming," pp. 43–49, 2018.

[25] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, "Measuring HTTP/3: Adoption and performance," pp. 1–8, 2021.

[26] D. Stenberg, "HTTP2 explained," pp. 120–128, 2014.

[27] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same standards, different decisions: A study of QUIC and HTTP/3 implementation diversity," pp. 14–20, 2020.

Accelerating QUIC with XDP

Minu Föger, Marcel Kempf *
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: foeger@in.tum.de, kempfm@net.in.tum.de

***Abstract*—After the standardization of the new transport protocol QUIC in 2021, QUIC implementations are increasingly optimizing for performance. Multiple implementations have started to utilize the eXpress Data Path (XDP) technology to speed up QUIC packet processing. XDP is a high-speed network data path in the Linux kernel based on the eBPF virtual machine. It allows bypassing the in-kernel network stack. A similar technique is being developed for Microsoft Windows. Due to the simplicity of UDP processing in QUIC, XDP shows great promise for reducing kernel overhead in QUIC implementations.**

***Index Terms*—Linux, XDP, QUIC**

1. Introduction

The transport protocol QUIC has seen increasing adoption on the Internet, with most modern browsers supporting QUIC [1]–[3] and with major web applications being serviced via HTTP/2-over-QUIC or HTTP/3 (e.g. most Google web applications) [4].

With the rising adoption of QUIC, major QUIC implementations have become more mature and are investing in performance optimizations. One potential optimization would be to use kernel bypass mechanisms to speed up lower-layer network packet processing. Most modern operating systems (like Linux or Windows) have generic in-kernel network stacks that usually provide transport layer abstractions (most prominently the Berkeley socket abstraction [5]). These network stacks are heavily optimized but are typically designed for high flexibility and generality. However, in specific cases this generality can result in inefficiencies [6].

QUIC is built on top of the old UDP protocol, which is typically implemented inside the kernel network stack (exposed via the socket API). However, QUIC is designed to require only minimal packet processing at lower layers, e.g. IP fragmentation is prohibited by the standard [7]. Therefore, it would be more efficient to perform QUIC-specific UDP payload extraction on incoming QUIC traffic than to process it via the generic kernel network stack. QUIC implementations would thus benefit from kernel bypass mechanisms. For this reason, multiple QUIC implementations (*s2n-quic* [8], *MsQuic* [9]) are currently starting to utilize the XDP kernel bypass mechanism to reduce the kernel overhead of their implementations.

There exist multiple techniques for high-speed packet processing that gain performance by bypassing the kernel network stack. One relatively new mechanism is the XDP/AF_XDP mechanism [6] that shall be further described in Section 2.

A popular alternative is DPDK (Data Plane Development Kit), a project facilitating high performance network I/O via specialized user-space Poll-Mode Drivers that poll the NIC directly and thus bypass the kernel network stack [10]. Initially developed by Intel in 2010 and targeting only the Linux kernel, it today is hosted collaboratively under the Linux Foundation and is also ported to other operating systems like Windows. DPDK has been optimized quite heavily over the years, often by taking advantage of hardware-specific features (especially on Intel hardware). Therefore, DPDK still remains able to achieve higher performance than XDP, though at the disadvantage of being highly invasive [11]–[13] (see also Section 2.6). There have been academic proposals to utilize DPDK to accelerate QUIC packet processing, e.g. *picoquic-dpdk*, which claims to have improved throughput by a factor of 3 compared to the original *picoquic* implementation [14].

There are also further kernel bypass mechanisms like PF_RING ZC, a proprietary module from the PF_RING project of ntop. PF_RING ZC is a zero copy packet processing framework that aims to provide a simple and hardware independent API and thus differs slightly in its abstraction level from the more low-level XDP and DPDK [15]. Since release 7.6.0 it has incorporated XDP and currently is built on top of AF_XDP by default [16].

2. XDP

The eXpress Data Path (XDP) is a subproject of the IO Visor Project hosted by the Linux Foundation, initially introduced in the Linux kernel in 2016 [18]. XDP is a high performance data path in the Linux kernel based on the eBPF runtime (extended Berkeley Packet Filter). It acts as an early hook in the Linux network receive (RX) path that can be used to bypass the kernel network stack [6]. XDP was initially designed to be a purely in-kernel mechanism for bypassing the network stack. A primary use case was DoS mitigation, i.e. the ability to drop packets from malicious traffic as early as possible. However, the development of the AF_XDP socket type in 2018 allowed XDP programs to redirect network packets efficiently to user space. This enabled the development of high performance networking applications in user space that bypass the kernel network stack completely [11].

 doi: 10.2313/NET-2024-04-1_03

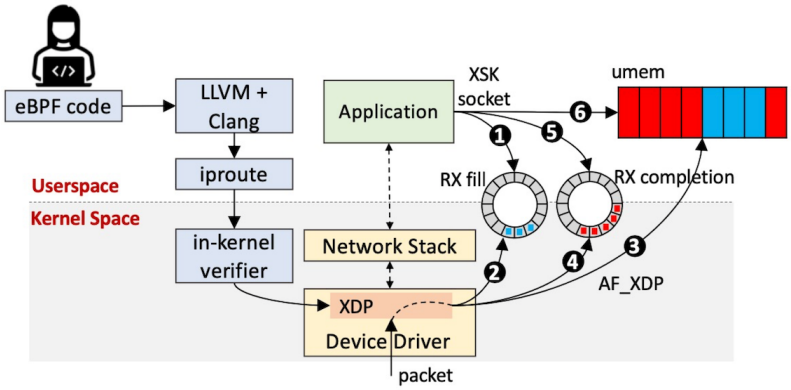


Figure 1: Receiving raw layer 2 frames via an AF_XDP socket (figure taken from [17])

2.1. eBPF

XDP builds on top of the extended Berkeley Packet Filter (eBPF), an extension to the largely deprecated classical Berkeley Packet Filter (cBPF) [5] in the Linux kernel. eBPF was initially released in 2014 (cBPF in 1992) and, like XDP, is part of the IO Visor Project. eBPF is an in-kernel virtual machine that allows to load and unload user-defined programs into designated code paths of the kernel network stack at runtime. The LLVM compiler backend supports compiling to eBPF bytecode. C (*libbpf* [19]) and Rust (*aya* [20]) libraries allow the programming of eBPF applications in higher level languages. eBPF programs have access to kernel networking infrastructure (e.g. routing tables) via BPF helper functions and are verified by the kernel on load to ensure program termination and to enforce in-kernel safety requirements.

The initial use case of BPF was to capture and filter network packets (e.g. by using tools like tcpdump). This is usually done via a tc (traffic control) eBPF program (type BPF_PROG_TYPE_SOCKET_FILTER) that hooks into either the ingress or egress point of the networking data path. Such eBPF programs are triggered before classical netfilter hooks, but after initial processing of the network stack (queuing, metadata extraction, GRO, etc.) as the hooks already reside inside the tc (traffic control) layer of the generic kernel network stack [21].

For high performance packet processing it is however more desirable to have an even earlier hook on the receive data path that circumvents the whole network stack. For such use cases the XDP program type was developed.

2.2. eBPF XDP Programs

An XDP program is a specific type of eBPF program (type BPF_PROG_TYPE_XDP). Once loaded on a network interface it is triggered directly after the networking driver has processed an incoming layer 2 frame from the NIC. It therefore bypasses the kernel network stack completely, runs before socket buffer allocation or GRO processing and operates on the raw layer 2 frame in the receive ring of the network driver (raw `xdp_buff` instead of metadata rich `sk_buff`). XDP programs can only be hooked to this ingress point of the networking data path and not to an egress point like with tc eBPF programs. The XDP program can arbitrarily process (and even modify) the received frame and eventually exits with a return value corresponding to a desired action to be triggered:

- **XDP_ABORTED**: drops the received frame and throws an eBPF tracepoint exception
- **XDP_DROP:** drops the received frame silently
- **XDP_PASS**: passes the received frame on to the in-kernel network stack
- **XDP_TX**: transmit the received frame back via the NIC it arrived on
- **XDP_REDIRECT**: redirect the received frame to another NIC or a user space AF_XDP socket

2.3. AF_XDP

AF_XDP is a specific socket type of the Linux Socket API that can be used to pass raw network packets efficiently between kernel space and user space via the fast XDP data path. The AF_XDP socket provides both receive and transmit semantics (though not via the classical syscalls). An AF_XDP socket (in user space) can receive raw layer 2 frames that a separate in-kernel XDP eBPF program has specifically redirected to it (via the XDP_REDIRECT action). This process is illustrated in Figure 1. An AF_XDP socket in user space can additionally transmit raw layer 2 frames out to the kernel that directly passes them to the networking driver. The socket type therefore bypasses the generic in-kernel network stack completely. The user-space application is thus fully responsible for parsing/processing the raw incoming layer 2 frames (received on the socket) and is also fully responsible for assembling the raw outgoing layer 2 frames (transmitted via the socket).

AF_XDP sockets are more complex in usage and configuration than regular socket types (e.g of type AF_INET). However, they enable high performance user-defined networking applications with similar performance to kernel bypass mechanisms that rely on hardware-specific user space drivers like DPDK. While AF_XDP provides a generic Linux interface that works regardless of hardware support, it only realize its full performance potential with driver support, e.g to facilitate zero copy semantics (see the XDP_ZEROCOPY flag) [21].

An AF_XDP socket is associated with the following data structures that users must configure and interact with:

- **UMEM**: A user-defined buffer of virtual contiguous memory consisting of equally sized frames. Network packets are passed between kernel space and user space via this buffer. The UMEM has two ring buffers associated with it:
  - **COMPLETION ring**: A consumer ring buffer in which the kernel stores pointers to UMEM frames when transmission of the corresponding frames has been completed.
  - **FILL ring**: A producer ring buffer in which the user space application can store pointers to UMEM frames that can be filled by the kernel when receiving on the socket.
- **TX ring**: A producer ring buffer in which the user space application can store pointers to UMEM frames that shall be transmitted by the kernel.
- **RX ring**: A consumer ring buffer in which the kernel stores pointers to UMEM frames that were received on the socket.

As with all sockets, a user application can wait for incoming packets or transmission completion by using the standard polling mechanisms of the Linux kernel (`poll()`, `select()`, `epoll()`). AF_XDP sockets can therefore be polled without pinning a whole CPU core to perform wasteful busy polling as was common in DPDK (however DPDK also supports event driven polling since version 17.05 [22]). Busy polling can still be performed with AF_XDP sockets via the `SO_BUSY_POLL` mechanism of the Linux socket API. This can be desirable, as busy polling ensures that AF_XDP applications can run using a single CPU core. Otherwise, two CPU cores will typically be used: one for the actual application and one for the RX/TX NAPI interrupt processing. This degrades performance as it causes costly coherency traffic between the cores [11].

2.4. XDP-for-Windows

XDP is a Linux technology, however Microsoft has also introduced an open source XDP implementation for the Windows operating system in 2022. This "XDP-for-Windows" implementation is heavily inspired but not a direct fork of the Linux XDP project [23].

As of September 2023, the implementation does not support eBPF programs, though the developers are claiming to aim for integration of XDP-for-Windows with the eBPF-for-Windows project. Instead of eBPF, XDP-for-Windows currently provides a custom built-in in-kernel XDP program. This program can not be coded directly by the user, but can be configured with matching rules and actions. This built-in XDP program can also be configured to redirect incoming network packets into user-space AF_XDP sockets (similar to the XDP_REDIRECT action in Linux XDP). The AF_XDP socket API is fully supported, but is not fully compatible with the Linux AF_XDP socket API. The underlying data structures used are analogous to Linux XDP, with a UMEM buffer and a TX, RX, COMPLETION, FILL ring buffer.

MsQuic, the QUIC implementation of Microsoft, already has experimental support for utilizing XDP-for-Windows in its Windows port [24].

2.5. Caveats of XDP

XDP is a mechanism to bypass the kernel network stack. Therefore, XDP can only benefit applications that do not require the functionality provided by the network stack (e.g. packet filtering for DoS mitigation) or can implement the functionality in a faster application-specific way (e.g. minimal UDP payload extraction in QUIC) [21]. Furthermore, an AF_XDP socket is not simply a high performance socket, but rather a way to pass raw layer 2 frames efficiently between the kernel and user space applications without having to traverse the kernel network stack. Unless applications can implement a more efficient way than the kernel network stack for processing raw layer 2 frames to the desired abstraction that classical sockets provide (e.g. a TCP byte stream), AF_XDP sockets can not replace classical socket usage in existing networking applications. When shortcutting the network stack via XDP, user applications also need to think about possible safety and security implications of their custom packet processing, e.g. if firewalls are bypassed, as the existing kernel networking mechanisms do not affect traffic redirected to AF_XDP sockets. For example, the QUIC implementations described in section 3 that utilize XDP bypass any classical firewalling on the system. Like all required functionality that would otherwise be provided by the kernel network stack, firewalling would have to be manually re-implemented when using XDP.

2.6. Comparison to DPDK

The main difference to other high speed packet processing mechanisms like DPDK is that XDP is highly integrated with the kernel. This means that XDP can provide an interface that is integrated into the standard operating system interfaces like eBPF or the socket API. XDP programs can also reuse existing networking capabilities of the kernel (e.g. routing tables). A major advantage compared to DPDK is that XDP facilitates device sharing, i.e. networking devices used by XDP applications remain visible/usable to non XDP applications [21]. XDP also does not require the use of huge pages like DPDK does [25]. A major drawback of XDP, however, is that it does not reach the performance of DPDK (e.g. 115 Mpps reached with DPDK against 100 Mpps reached with XDP in a packet drop benchmark on five cores [6]). CPU usage of DPDK was worse than XDP in the past due to DPDK requiring busy polling, but DPDK has since added support for event driven polling. Overall, XDP seems to be better suited in use cases that require minimal invasiveness and are more hardware agnostic (e.g. classical desktop applications), whereas DPDK seems to be better suited for specialized use cases that can tolerate high invasive measurements to achieve maximal performance (e.g. the classical DPDK use case of accelerating telecom NFV).

3. QUIC Implementations

The QUIC protocol is designed to replace the widely used TCP/TLS stack for web traffic. The proposed HTTP/3 standard is explicitly implemented on top of QUIC [26]. The protocol was standardized in 2021 with RFC 9000 [7]. QUIC implementations are still under

active development. But as QUIC is increasingly being adopted on the Internet, they are becoming ever more mature and are starting to invest more into optimizing performance. Major QUIC implementations are currently starting to utilize kernel bypass mechanisms to speed up their implementations, in particular XDP.

3.1. Concept of XDP Usage in QUIC

Conceptually, XDP is utilized in QUIC implementations to speed up UDP payload extraction from sockets. The QUIC protocol is designed to be implemented in user space instead of inside the kernel (where transport layer protocols typically are implemented) to allow for quicker and more flexible development cycles. However, the QUIC standard mandates a minimal UDP layer underneath QUIC, mainly to enable the traversal of legacy middleboxes [7].

A lot of the functionality of the generic kernel network stack is not necessary to process incoming QUIC traffic. Due to a combination of restrictions in the QUIC standard and its underlying network stack as well as the behavior of modern NICs, UDP payloads can be easily extracted from incoming layer 2 frames.

- **Layer 2:**
  The layer 2 protocol can be assumed to be Ethernet, since most modern NICs (or their corresponding drivers) will always emit incoming frames in ethernet format (even if it is received via WLAN it will wrap the received 802.11 payload into a "fake" ethernet frame). MAC addresses also do not need to be checked as networking devices are not set to promiscuous mode by default. Payloads are extracted by simply removing the Ethernet header.
- **Layer 3:**
  The layer 3 protocol used almost exclusively on the Internet is either IPv4 or IPv6. The QUIC standard forbids IP fragmentation by design [7]. This simplifies payload extraction, since one raw ethernet frame always holds exactly one UDP datagram. The IP header only needs to be processed minimally: IP addresses and the ECN (explicit congestion notification) must be stored for processing in higher layers of QUIC (e.g. connection migration and congestion control). The IPv4 checksum is not checked to improve performance. This can be assumed to be safe, since stronger integrity checks are employed in both lower layers (e.g. via ethernet CRCs) and higher layers (e.g. via TLS MAC/AEAD) of the employed QUIC network stack. IPv6 does not employ a checksum field for similar reasons [27]. Other fields of the IP header are also ignored by the examined implementations. Payloads are extracted by simply removing the IP header.
- **Layer 4:**
  Although QUIC itself implements layer 4 functionality, it builds on top of UDP by design. UDP is a minimalistic protocol that requires minimal processing. The ports are noted for processing in the QUIC connection migration mechanism. The UDP checksum is not checked to improve performance. This can be assumed to be safe for the same reasons as with IPv4 checksums. Payloads are extracted by simply removing the UDP header.

These steps can be done via XDP to process the incoming QUIC traffic on the RX path manually up to the UDP layer, bypassing the kernel network stack. For the TX path the process is inverted, however, all fields must be filled (even those ignored on the RX path) and all checksums must be computed.

3.2. Specific QUIC Implementations

To the best of the author's knowledge, currently two major QUIC implementations exist that utilize XDP:

- ***MsQuic* from Microsoft [9]:**
  The XDP implementation of *MsQuic* is currently (September 2023) under active development. It is released under "preview support" for the Windows porting of *MsQuic* and uses XDP-for-Windows (Linux is not supported yet) [24]. *MsQuic* is implemented in C++ and can therefore directly use the native XDP-for-Windows libraries. The implementation configures the built-in XDP program of XDP-for-Windows to redirect incoming traffic to AF_XDP sockets. The manual packet processing of UDP frames as described above is performed completely in user space. Currently the application supports only busy polling on AF_XDP sockets. Microsoft claims that by utilizing XDP it can improve both latency and throughput by ca. 100% on serialized HTTP requests [28].
- ***s2n-quic* from Amazon AWS [8]:**
  The XDP implementation of *s2n-quic* is currently (September 2023) also under active development and only available for Linux. *s2n-quic* is implemented in Rust and uses the *aya* eBPF/XDP library. It must wrap Linux C syscalls for use in Rust. The implementation uses a custom in-kernel eBPF XDP program to redirect traffic destined to the correct UDP port into AF_XDP sockets. The manual packet processing of UDP frames as described above is then performed in user space. *s2n-quic* supports both busy polling and event-based polling (via the *tokio* runtime) on AF_XDP sockets. XDP is not yet fully integrated into the API, but *s2n-quic* does provide a full server-client example ("s2n-quic-qns") that can be configured to use XDP. AWS does not provide detailed performance analysis but released profiling data indicates a significant reduction of kernel overhead when using XDP [29].

In addition, there has been an attempt by LiteSpeed Technologies to add XDP support to their *lsquic* QUIC implementation [30]. The design of the implementation is similar to *s2n-quic*: it utilizes Linux XDP with a separate eBPF XDP program that efficiently extracts UDP payloads and redirects to AF_XDP sockets. LiteSpeed claims that this gave a 43% improvement in throughput compared to standard UDP sockets [31]. However, this was only implemented as a proof-of-concept and was never incorporated into the official *lsquic* implementation.

4. Conclusion and future work

Kernel bypass mechanisms for networking applications are important tools to facilitate high-performance packet processing on general purpose processors/operating systems. XDP is an upcoming technology of this kind. It allows bypassing the kernel network stack and optionally passing raw layer 2 frames efficiently between user space applications and the kernel. Although it is still slower than its main competitor DPDK, it has the advantage of being highly integrated with the kernel (e.g. it does not require user space applications to take full control of the NIC).

Major implementations of the new QUIC transport protocol have started to utilize the XDP technology to speed up their implementations by reducing the kernel overhead of the standard UDP socket API. Due to the specifics of the QUIC protocol and its underlying network stack, the extraction of QUIC payloads from its underlying UDP layer can be done in a minimalistic manner that is more efficient than the generic kernel network stack. XDP can leverage this fact and has shown promising results for improving performance in major QUIC implementations.

References

[1] I. S. David Schinazi, Fan Yang, "Chrome is deploying HTTP/3 and IETF QUIC," 2020, accessed: 2023-09-30. [Online]. Available: https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html

[2] D. Damjanovic, "QUIC and HTTP/3 Support now in Firefox Nightly and Beta," 2021, accessed: 2023-09-30. [Online]. Available: https://hacks.mozilla.org/2021/04/quic-and-http-3-support-now-in-firefox-nightly-and-beta

[3] Apple Inc., "Safari 14 Release Notes," 2020, accessed: 2023-09-30. [Online]. Available: https://developer.apple.com/documentation/safari-release-notes/safari-14-release-notes

[4] E. Sy, C. Burkert, H. Federrath, and M. Fischer, "A QUIC Look at Web Tracking." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 255–266, 2019.

[5] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture." in *USENIX winter*, vol. 46, 1993, pp. 259–270.

[6] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *Proceedings of the 14th international conference on emerging networking experiments and technologies*, 2018, pp. 54–66.

[7] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[8] Amazon.com, Inc, "s2n-quic Github," accessed: 2023-09-30. [Online]. Available: https://github.com/aws/s2n-quic

[9] Microsoft Corporation, "MsQuic Github," accessed: 2023-09-30. [Online]. Available: https://github.com/microsoft/msquic

[10] A. LLC, "DPDK White Paper: Myth-Busting DPDK in 2020," The Linux Foundation, Tech. Rep., 2020.

[11] M. Karlsson and B. Töpel, "The path to DPDK speeds for AF_XDP," in *Linux Plumbers Conference*, 2018.

[12] E. Freitas, A. T. de Oliveira Filho, P. R. do Carmo, D. F. Sadok, and J. Kelner, "Takeaways from an experimental evaluation of eXpress Data Path (XDP) and Data Plane Development Kit (DPDK) under a Cloud Computing environment," *Research, Society and Development*, vol. 11, no. 12, pp. e26 111 234 200–e26 111 234 200, 2022.

[13] J. D. Brouer and T. Høiland-Jørgensen, "XDP: challenges and future work," in *Proc. Linux Plumbers Conference*, 2018.

[14] N. Tyunyayev, M. Piraux, O. Bonaventure, and T. Barbette, "A high-speed QUIC implementation," in *Proceedings of the 3rd International CoNEXT Student Workshop*, 2022, pp. 20–22.

[15] A. Cardigliano, "Positioning PF_RING ZC vs DPDK," 2017, accessed: 2023-09-30. [Online]. Available: https://www.ntop.org/pf_ring/positioning-pf_ring-zc-vs-dpdk/

[16] ntop, "PF_RING 7.6.0 release," 2020, accessed: 2023-09-30. [Online]. Available: https://github.com/ntop/PF_RING/releases/tag/7.6.0

[17] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vswitch dataplane ten years later," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 245–257.

[18] IO Visor Project, "XDP: eXpress Data Path," accessed: 2023-09-30. [Online]. Available: https://www.iovisor.org/technology/xdp

[19] Linux Project, "libbpf Github," accessed: 2023-09-30. [Online]. Available: https://github.com/libbpf/libbpf

[20] Aya Project, "aya Github," accessed: 2023-09-30. [Online]. Available: https://github.com/aya-rs/aya

[21] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[22] "DPDK Release 17.05," 2017, accessed: 2023-09-30. [Online]. Available: https://doc.dpdk.org/guides/rel_notes/release_17_05.html

[23] XDP-for-Windows, "Frequently Asked Questions," accessed: 2023-09-30. [Online]. Available: https://github.com/microsoft/xdp-for-windows/blob/main/docs/faq.md

[24] Microsoft Corporation, "MsQuic v2.2.0," 2018, accessed: 2023-09-30. [Online]. Available: https://github.com/microsoft/msquic/releases/tag/v2.2.0

[25] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Accelerating virtual network functions with fast-slow path architecture using express data path," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1474–1486, 2020.

[26] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[27] D. S. E. Deering and B. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 8200, Jul. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8200

[28] Y. Huang, "Balance Performance in MsQuic and XDP," 2022, accessed: 2023-09-30. [Online]. Available: https://techcommunity.microsoft.com/t5/networking-blog/balance-performance-in-msquic-and-xdp/ba-p/3627665

[29] C. Bytheway, "PR: feat(s2n-quic-qns): add XDP server #1765," 2023, accessed: 2023-09-30. [Online]. Available: https://github.com/aws/s2n-quic/pull/1765

[30] LiteSpeed Technologies Inc, "lsquic github," accessed: 2023-09-30. [Online]. Available: https://github.com/litespeedtech/lsquic

[31] R. Perper, "Performance Comparison of QUIC with UDP and XDP," 2022, accessed: 2023-09-30. [Online]. Available: https://blog.litespeedtech.com/2020/06/01/performance-comparison-quic-udp-xdp

Extracting Information from Machine Learning Models

Iheb Ghanmi, Lars Wüstrich*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: iheb.ghanmi@tum.de, wuestrich@net.in.tum.de

***Abstract*—In the era of data-driven decision-making, machine learning models, especially neural networks, demonstrate their capabilities across various domains. However, as the deployment of these models increases, the vulnerability of these models to attacks has become a significant concern. This paper illustrates how attackers can extract sensitive information from machine learning models, potentially compromising the confidentiality of training data. We introduce the fundamentals of neural networks, emphasize the architecture of Feedforward Neural Networks, and explain how weights and biases intrinsically store knowledge. We present two attacks designed to extract information about the training data from a black-box neural network.**

***Index Terms*—neural networks, adversarial attacks, information extraction, privacy**

1. Introduction

Machine learning (ML) currently undergoes a transformative evolution. Transitioning from an academic curiosity, it now serves as a pivotal tool in numerous real-world applications [1]. Present capabilities include detecting patterns in images [2], decoding nuances of human language [3], and recognizing intricate auditory cues [4]. Typically, these models are encapsulated and operate as "black-boxes". In this configuration, users access the input-output relationships but remain uninformed about internal operations [5]. This design choice simplifies user interactions and, crucially, safeguards sensitive data included in the training datasets [6].

However, the perceived simplicity and security hide inherent challenges. Our review reveals that over-reliance on the obscured nature of black-box models for security is problematic. Despite their strengths, neural networks (NNs) are susceptible to attacks that aim to reveal information, particularly regarding the data they were trained on [7]. Most crucially, and as our primary contribution, we examine various techniques, shedding light on the **inherent weaknesses** of these networks, challenging their perceived *invulnerability*, and underscoring the **urgent** need for better defenses [8].

The remainder of this paper is structured as follows: Section 2 introduces the basics of NNs, Section 3 defines our threat model, and Section 4 delves into methods for information extraction. In Section 5, we explore specific applications of the attacks within the networking domain, especially in Intrusion Detection Systems (IDS). We conclude with a discussion on future directions.

2. NN Basics

This section introduces concepts related to NNs that are essential for understanding subsequent discussions for attacks and defenses. An NN is a computational model inspired by biological NNs in the human brain. The primary function of an NN is to receive input, process it, and provide an output.

2.1. NN Architecture

The architecture of an NN defines its fundamental structure, detailing how individual components, such as neurons, are interconnected. This structure plays a pivotal role in determining the network's computational capabilities and its ability to learn from data.

Neurons: Neurons are fundamental units in an NN. A neuron receives multiple inputs, processes them, and generates a single output. This processing involves a **weighted** sum of the inputs, an addition of a **bias**, and the application of an activation function [9].

Activation Functions: These are mathematical functions that, given an input, determine the output of a neuron. Common activation functions include the sigmoid, tanh [10], and ReLU (Rectified Linear Unit) [11].

Layers: Typically, we organize NNs in layers [9]. The three main types of layers are:

Input Layer: This is where the network receives input from the dataset. Each neuron in this layer corresponds to one feature in the dataset.

Hidden Layer: These are layers between the input and output layers and are each composed of multiple neurons. An NN can have any number of hidden layers, and this is what makes a network "deep" in deep learning [12].

Output Layer: This layer produces the final prediction or classification of the network.

Feedforward NNs (FNNs): FNNs represent the most straightforward artificial NN architecture type [10]. In FNNs, the data flows in one direction, from the input layer, through the hidden layers, and to the output layer. There are no cycles or loops in the network. Figure 1 provides an overview of such a network.

2.2. Knowledge in NNs

NNs store knowledge as weights and biases. Weights determine the connection strength between two neurons. Biases, similar to intercepts in linear equations, allow neuron output adjustments. During training, the network

 doi: 10.2313/NET-2024-04-1_04

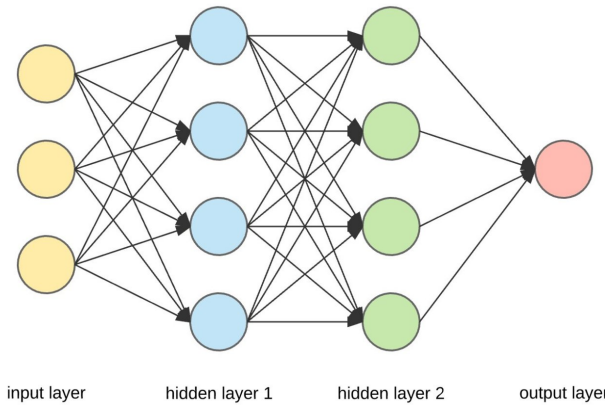


Figure 1: Overview of a simple example structure of an FNN of one input layer, two different hidden layers and one output layer consisting of one neuron. Model was adapted from [13]

modifies its weights and biases to reduce prediction discrepancies from actual outcomes, typically using backpropagation and optimization techniques like gradient descent [9]. A common challenge with NNs is their black-box nature. Although they produce relatively accurate predictions, explaining the exact reasoning behind specific decisions remains difficult [14]. The knowledge in the network is distributed across the NN's weights and biases, and it is not always clear how individual weights contribute to the final decision.

3. Threat Model

This section defines our threat model which consists of the properties of our target model as well as the attacker's capabilities.

Target Model: The target model refers to any trained machine learning model, particularly those that have been trained on sensitive datasets, such as medical records or personal information. These models range from deep NNs [15] to classification models trained by popular "machine learning as a service" providers [16].

Attacker Capabilities:
* **Model Access:** The attacker can access the model, meaning they can send input data to the model and receive the corresponding outputs. This does not imply that the attacker has access to the original training data or any metadata associated with it.
* **Input Data:** The attacker can provide any input data to the model and observe its predictions or classifications. This allows the adversary to infer information about the model's training data or its internal workings.
* **Model Queries:** The attacker can make unlimited queries to the model. This means that the attacker can send an unlimited number of input data points to the model and observe the corresponding outputs.

Attacker Limitations:
* **Black-box Assumption:** Although the attacker interacts with the model, they do not have direct access to the model's internal parameters, weights, or architecture. This means that the attacker cannot directly observe or manipulate the inner workings of the model.
* **No Training Data Access:** The attacker does not have access to the original training data or any associated metadata. This is particularly relevant in scenarios where the training data is sensitive or confidential.

4. Information Extraction Methods

This section focuses on attacks that aim to extract information about the training data of the NNs. The goal is to analyze and compare the methods that were proposed in the literature. The discussion starts with introducing the different methods and then comparing them in terms of their efficiency and accuracy. The section also discusses the different assumptions that were made by the authors of the different methods.

In this paper, the focus is on the following methods: Membership Inference Attacks in Section 4.1 and Knowledge Extraction Attacks with No Observable Data in Section 4.2. These represent two possible methods to extract information about the training data.

4.1. Membership Inference Attacks (MIAs)

Membership Inference is not a singular adversarial attack but rather represents a broader category of such attacks. In these attacks, the objective is to determine whether a specific data point belongs to the training dataset by interacting with a model via a black-box method. This method circumvents the need to rely on explicit statistics or specific details about the target model's architecture. Through this technique, the attacker learns actual information about whether a specific data point is part of the model's original training dataset.

In this context, the discussion revolves around the attack methodology introduced by Shokri et al. [16]. Here, the authors trained an **attack model** to distinguish the target model's responses based on whether the input data is part of its original training dataset (see Figure 2).

Membership inference attacks, as explored by Shokri et al. [16], utilize a technique termed shadow training. In this approach, multiple **shadow models** are constructed to mimic the behavior of the target model. These shadow models are trained on datasets that closely resemble the distribution of the target model's training data (Figure 3). A salient feature of these shadow models is the attacker's awareness of their training datasets, ensuring a clear understanding of data record membership.

This knowledge facilitates the training of the attack model using the input-output pairs from these shadow models. While various strategies can be employed to generate this data, it's paramount that the data distribution aligns closely with that of the target model's training set. By contrasting the behavior of the shadow models on their known training data with their behavior on unfamiliar inputs, the attack model can discern nuanced differences in the target model's responses.

The study highlights that, even without prior assumptions about the distribution of the target model's training data, and using fully synthetic data for shadow models, membership inference accuracy can reach up to 90% [16]. Furthermore, the research underscores the potential risks to datasets, such as those from health care, when used to

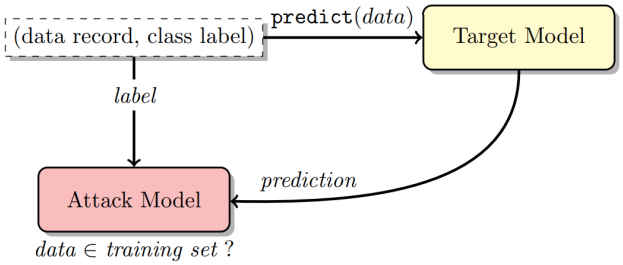


Figure 2: Overview of the Membership Inference Attack adapted from [16]. The *attack model* receives, along with the class label of the input, the prediction output of the original *target model*. A classification is then made to ascertain whether the input data was part of the original training dataset.

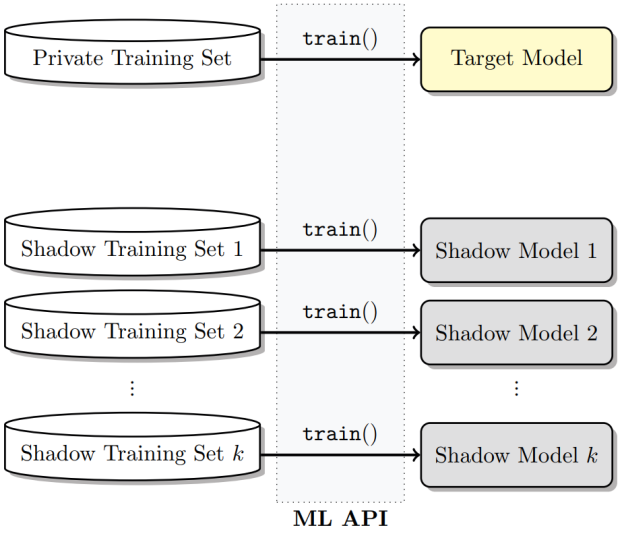


Figure 3: Overview of the training of *shadow models*adapted from [16]. *Shadow training sets* are constructed and used to train each of the models separately. The datasets share the same format but contain different data points from similar distributions.

train machine learning models that are publicly accessible. This method lets an attacker reconstruct the training data, making the potentially sensitive information available to them. The efficacy of this method is contingent upon the number and caliber of the shadow models and the congruence of their training datasets with the target model's dataset. For a detailed understanding of the specific training methodology employed by Shokri et al. for the shadow models, readers are referred to the original publication.

4.2. Knowledge Extraction Attacks with No Observable Data

In the paper titled "Knowledge Extraction with No Observable Data" [15], Yoo et al. present methods for two scenarios: available and hidden training data. NNs are parameterized functions designed to approximate arbitrary functions, specified by training data examples. The architecture of the network defines its computational structure, while the parameters or weights determine its specific

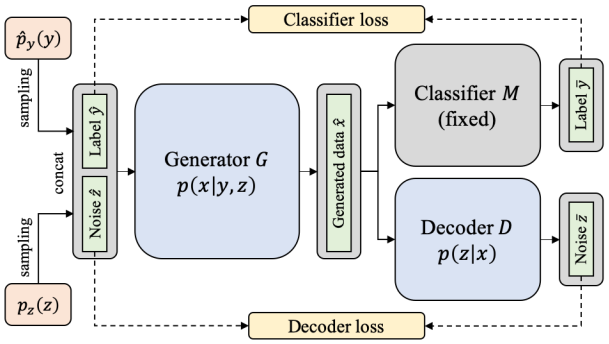


Figure 4: Overview of the KEGNET's operation adapted from [15]. The generator network G utilizes sampled variables $\hat{y}$ and $\hat{z}$ to produce a fake data point $\hat{x}$, aiming to mimic the original training data distribution by minimizing the Kullback-Leibler (KL) divergence. Concurrently, the decoder network D strives to retrieve the variable $\hat{z}$ from $\hat{x}$ and reconstruct the original input, minimizing the mean squared error (MSE) between the original and reconstructed data. Both networks undergo end-to-end training: G generates data points fed into a fixed classifier and D, while D extracts a low-dimensional representation. The iterative training refines both networks based on discrepancies between generated and original data and between original and reconstructed inputs.

computations. The paper introduces the concept of "unintended memorization", where NNs might inadvertently reveal out-of-distribution training data, termed as "secrets". From this method, the attacker extracts information stored within the NN itself. The exact nature of this information, such as the numbers depicted in Figure 5, is open to the interpretation of the attacker.

For scenarios with available data, Yoo et al. [15] introduce KEGNET (Knowledge Extraction with Generative Networks). This method aims to move knowledge from a large NN (known as the *teacher network*) to a smaller one (called the *student network*). The authors designed KEGNET, especially for scenarios where there is not much training data or the student model needs to be small. Figure 4 shows a visual explanation of the KEGNET process.

However, when the original training data is concealed, especially in areas such as medicine and defense, the challenges increase. To solve this, KEGNET uses tools to create fake data points that can replace the hidden original training data. The main idea here is that the process of pulling out knowledge focuses on a small set of data points within a certain area [15]. This led to the creation of a generator network, paired with a discriminator network, to mimic and tell apart data points.

The team tested KEGNET on three datasets from the UCI Machine Learning Repository. Using a multilayer perceptron as a classifier and adding Tucker decomposition to all dense layers, the results showed KEGNET did better than other standard methods. This means an attacker can use KEGNET to pull knowledge from different NN designs and various types of training data.

While the main paper shows possible weak points in hidden models, it does not give a clear method to

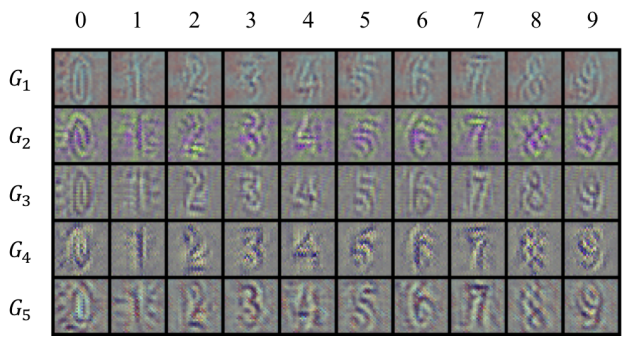


Figure 5: Visual representation of the artificial data points generated by the generator network of KEGNET [15]. These data points exemplify how the generator can produce synthetic data that closely resembles the original training data. The original teacher network was trained on the SVHN dataset [17]. Figure adapted from [15].

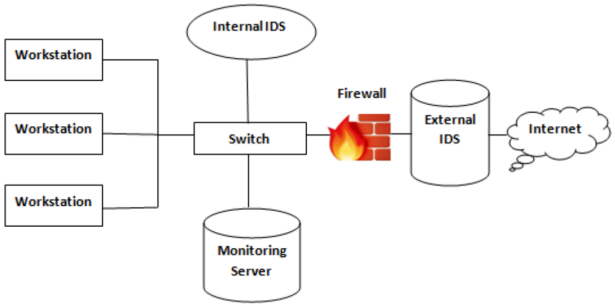


Figure 6: A schematic representation of an IDS showcasing its core components and their interactions within a network environment. This model emphasizes the critical role of IDSs in monitoring network activities, verifying connection patterns, and analyzing the flow of packets to detect potential threats. Adapted from [18].

pull out specific details about the training data. However, the authors do mention the generator network's ability to create data points similar to the original training data. An example of this can be seen in Figure 5.

In conclusion, Yoo and his team introduce a significant advancement in machine learning by crafting a mechanism that extracts knowledge sans observable data. Their method, KEGNET, offers a solution for scenarios constrained by data accessibility due to privacy or confidentiality nuances. KEGNET provides also an opportunity for attackers to directly extract secrets stored in an NN.

5. Applications in the Networks Domain

In this section, we explore a specific application of one of the attacks (MIA) introduced in Section 4 within the networking domain. A prime example where NNs are extensively used is in Intrusion Detection Systems (IDSs).

5.1. Intrusion Detection Systems

IDSs serve as a cornerstone in the realm of network security, vigilantly monitoring network traffic to detect malicious activities [18]. A pivotal aspect of their operation hinges on the training data, which predominantly consists of network logs [18]. Figure 6 provides a visual representation of the core components and functioning of an IDS.

5.2. Learning Mechanisms of IDS

IDSs derive their efficacy from extensive training on network logs. These logs capture diverse network activities, protocols, and communication patterns. By assimilating this data, IDSs not only recognize but also learn the underlying patterns of regular and anomalous traffic. This learned knowledge empowers them to swiftly identify and respond to potential threats, ensuring robust network security [18].

5.3. Vulnerabilities and Potential Attacks on IDS

IDSs are not impervious to threats. One of the most potent threats they face is MIAs. These attacks are designed to reverse-engineer the data on which the IDS was trained. By successfully executing an MIA, attackers can gain information from the system. For instance, they can:
* **Pinpoint commonly used services:** By analyzing the network logs, attackers can identify frequently used ports, such as port 53, which is typically associated with DNS.
* **Determine entities running specific services:** Through MIAs, attackers can discern which specific nodes or entities within the network are responsible for running certain services, like DNS servers.
* **Extract overarching network structure insights:** Beyond just services, MIAs can provide attackers with a broader understanding of the network's layout, its key entities, and their interrelationships.

5.4. Speculative Implications of MIAs on IDS

The implications of successful MIAs on IDS are not just limited to information extraction. Armed with the knowledge obtained from MIAs, skilled attackers can craft malicious packets that blend seamlessly with regular traffic, evading detection by the IDS. This potential scenario emphasizes the urgent need for enhanced defenses against such sophisticated attacks.

Countermeasures: While the primary focus of this paper is on the vulnerabilities, it is worth noting that the research community is not standing still. Efforts are being made to develop defense mechanisms against such attacks, as seen in [19], which proposes defense strategies based on gradient differential privacy.

6. Conclusion and future work

In this paper, we described the intricate manner in which NNs store information. Subsequently, we introduced two distinct methods for knowledge extraction from NNs when presented as a black box. In the application of one of these attacks to the networking domain, we demonstrated their significant implications, particularly within Intrusion Detection Systems.

Moving forward, more research is imperative on robust defense mechanisms, interdisciplinary collaboration between machine learning and cybersecurity, and the development of transparent and interpretable NN architectures.

Our findings underscore the importance of these areas in ensuring the security and efficacy of NNs in real-world applications.

References

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[4] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[5] W. J. von Eschenbach, "Transparency and the black box problem: Why we do not trust ai," *Philosophy & Technology*, vol. 34, no. 4, pp. 1607–1622, 2021.

[6] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[7] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 601–618. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer

[8] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[9] J. A. Anderson, *An introduction to neural networks*. MIT press, 1995.

[10] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 39, no. 1, pp. 43–62, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169743997000610

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] A. Rastogi, K. Agarwal, E. Lolon, M. Mayerhofer, and O. Oduba, "Demystifying data-driven neural networks for multivariate production analysis," in *Unconventional Resources Technology Conference, Denver, Colorado, 22-24 July 2019*. Unconventional Resources Technology Conference (URTeC); Society of . . . , 2019, pp. 2602–2622.

[14] D. Castelvecchi, "Can we open the black box of ai?" *Nature News*, vol. 538, no. 7623, p. 20, 2016.

[15] J. Yoo, M. Cho, T. Kim, and U. Kang, "Knowledge extraction with no observable data," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/596f713f9a7376fe90a62abaaedecc2d-Paper.pdf

[16] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[18] N. Unnisa A, M. Yerva, and K. M Z, "Review on intrusion detection system (ids) for network security using machine learning algorithms," *International Research Journal on Advanced Science Hub*, vol. 4, no. 03, pp. 67–74, 2022. [Online]. Available: https://rspsciencehub.com/article_17618.html

[19] Z. Liu, R. Li, D. Miao, L. Ren, and Y. Zhao, "Membership inference defense in distributed federated learning based on gradient differential privacy and trust domain division mechanisms," *Security and Communication Networks*, vol. 2022.

Probabilistic Network Telemetry

Benjamin Schaible, Kilian Holzinger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: b.schaible@tum.de, holzinger@net.in.tum.de

***Abstract*—With ever increasing demand for large scale computing systems such as data centers, high-speed interconnecting networks too are becoming increasingly important. Measuring these networks is key to detecting and localizing performance issues. One important metric is packet latency. However, established network measurement systems are either not suitable for estimating packet latencies or introduce unwanted overhead [1].**

In this paper, we discuss three probabilistic approaches to estimating packet latencies, which were proposed in prior work: the Lossy Difference Aggregator (LDA), the Lossy Difference Sketch (LDS) and Reference Latency Interpolation (RLI). Following that, we compare these approaches against each other regarding their robustness to packet loss and reordering, their accuracy and the overhead they introduce. We come to the conclusion that, under low packet loss, latency estimates provided by the LDA and LDS are more accurate than ones provided by RLI.

***Index Terms*—probabilistic data structures, network measurement, high-speed networks**

1. Introduction

With ever increasing demand for large scale computing systems such as data centers, high-speed interconnecting networks too are becoming increasingly important. Such networks may consist of thousands of devices, switches and routers, making it nearly impossible to predict unwanted behavior. Network measurements have therefore become an essential tool to detect and localize network performance problems such as bottlenecks.

One of the key metrics measured is packet latency [1]–[3], which is the amount of time it takes for a packet to travel from one point to another in a network. Efficiently measuring packet latencies, however, especially in the order of microseconds, is a non-trivial matter. Established systems like NetFlow actively measure packet latencies by attaching timestamps to a sample of all observed packets [1]. High sampling rates are required in order for this to yield accurate average latency estimates, which in turn imposes a considerable network overhead [1].

In this paper, we discuss three probabilistic approaches to estimating average packet latencies, which were proposed in priror work. These approaches aim to provide accurate average latency estimates while remaning lightweight in terms of network utilization, memory usage and computational overhead. Namely, we discuss the Lossy Difference Aggregator [1] in Section 3.1, the Lossy Difference Sketch [2] in Section 3.2 and Reference Latency Interpolation [3] in Section 3.3. Following that, we compare their strengths and weaknesses in Section 4.

2. Probabilistic Data Structures

All of the probabilistic measurement approaches presented in this paper leverage their own application-specific data structures, though they share the same core concepts. We now discuss the most important ideas behind the probabilistic data structures used.

The data structures we are going to discuss in this paper are *sketches*. Sketches are used to *estimate* various metrics on streams of data. While they don't guarantee exact results, they usually come with a much smaller overhead when compared to perfectly precise measurements. Instead, they are *likely* to yield accurate results and therefore are being referred to as probabilistic.

A sketch usually aggregates some kind of value in a counter. In our case these values would be packet latencies. However, there usually is an unlikely but catastrophic occurrence which would *destroy* the counter, meaning that the value stored is skewed heavily and can no longer be used to obtain accurate estimates. In our case, this occurrence would be packet loss.

To deal with such occurrences, sketches use multiple counters and distribute samples evenly across them. This way, such a catastrophic occurrence would only *destroy* one of many counters, leaving the rest of them in a usable state. This even distribution is commonly achieved by calculating a hash of the sample and mapping that hash to an index within an array of counters. For example, assume that our sketch consists of n counters and we wish to insert the latency of a packet p. We may then calculate an index with $hash(p) \mod n$. Assuming the hash function used is *uniform*, meaning that each hash is produced with the same probability, this would result in an even distribution of samples across all n counters. In this paper, we refer to this technique as *hash partitioning*.

3. Measurement Approaches

Consider the following example: Two clients are exchanging a considerable amount of data via two switches (Figure 1). Now, let's say we want to measure the latency (or "delay") between these two switches, for example in order to gain insight into the state of the network. A simple way to achieve this is to have both switches attach a timestamp to each packet before

 doi: 10.2313/NET-2024-04-1_05

dispatching it. The receiving switch may then calculate the latency by subtracting the packet's timestamp from its current time. This also allows one to calculate the average latency by summing up all measured delays and dividing by the number of delays measured.

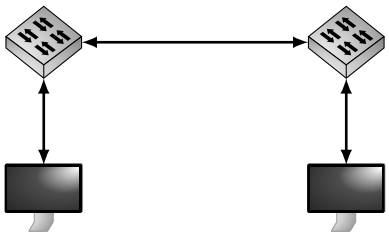


Figure 1: A simple measurement environment

The main problem with this approach is the overhead in bandwidth usage caused by the timestamps attached to each packet. Especially in data center networks, where hundreds of millions of packets are transmitted every second, even relatively small (32-bit) timestamps would quickly add up.

Additionally, any difference between clocks of the two switches will directly add to the error of the measured latency [1]. Therefore, both switches's clocks have to be synchronized tightly in order for the measurement to yield accurate results [1]. We are assuming microsecond-level clock synchronization, which may be achieved by making use of protocols such as IEEE-1588 [4].

We now discuss several probabilistic approaches to estimating the average latency between two network points.

3.1. Lossy Difference Aggregator

The Lossy Difference Aggregator (LDA), proposed by Kompella et al. in [1], is a probabilistic data structure, used to efficiently estimate the average and standard deviation of the latency between a sender A and a receiver B. The sender and receiver may be two network devices such as routers or switches, or two points within a single network device [1].

An LDA is used to conduct a *one-way* measurement, meaning that it estimates the average latency of packets being sent from A to B, ignoring packets sent the other way [1]. If a bi-directional measurement is desired, the measurement should be conducted twice, once in each direction.

3.1.1. Basic Idea. We now discuss the basic approach behind the LDA. Assume that A sends a total of N packets to B and we wish to measure the average latency of these packets. Let a_i be the time at which the i-th packet was dispatched from A, and b_i the time it was received at B, with $i \in \{1, \ldots, N\}$. For now, we assume there is no packet loss. Consequently, the latency of the i-th packet is $b_i - a_i$ and the average latency D corresponds to

$$D = \frac{1}{N} \sum_{i=1}^{N} (b_i - a_i) \tag{1}$$

$$= \frac{1}{N} \left(\sum_{i=1}^{N} b_i - \sum_{i=1}^{N} a_i \right) \tag{2}$$

as described in [1].

Recall the previously discussed approach of attaching a timestamp to each packet. In this context, A would attach to each packet its timestamp a_i, while B would count the number of packets it receives and add each packet's delay $b_i - a_i$ to an accumulator. The average delay could then be calculated using (1).

A Lossy Difference Aggregator allows one to conduct this type of measurement without the overhead of attaching timestamps to any packet. It operates on periodic time intervals of length T, with T usually being in the range from a few hundred milliseconds to a few seconds [1]. When using an LDA, instead of attaching timestamps to packets, we aggregate them both on the sender and receiver [1]. Then, at the end of each measurement interval, a control packet containing the sender's timestamp aggregate is sent to the receiver and used to calculate the average latency there [1]. The LDA is the data structure used to aggregate those timestamps, both at the sender and receiver [1].

In the context of the current example, consider a simplified LDA consisting of a single timestamp aggregator and a packet counter. We use one simplified LDA each for the sender and receiver. At the sender A, the simplified LDA then stores the sum $T_S = \sum_{i=1}^{N} a_i$ and the amount of packets sent N. At the receiver B, it stores the sum $T_R = \sum_{i=1}^{R} b_i$ and the amount of packets received R. Since we assume no packet loss, N and R are equal. As shown in [1], we can then transform (2) to use T_S and T_R:

$$D = \frac{1}{N} (T_R - T_S) \tag{3}$$

Using (3), we are able to calculate the average latency during a measurement cycle by building timestamp aggregates T_S and T_R at the sender and receiver respectively and by transmitting T_S to the receiver at the end of the cycle, assuming that $R = N$.

3.1.2. Dealing with Packet Loss. The simplified LDA however has a significant flaw: If even a single packet sent from A to B is lost, meaning that $N > R$, the receiver's timestamp accumulator becomes *unusable* for the remainder of the measurement interval and no meaningful average delay can be calculated. This is due to the fact that T_S would have aggregated more timestamps than T_R and that it is impossible to recover the original timestamps aggregated in T_S and T_R, eliminating the possibility of calculating the average of a smaller sample [1].

It is possible to mitigate the severity of this problem by making use of a simple measure proposed in [1]: Instead of a single accumulator-counter pair, use an array of m of these pairs, which we call a *bank* [1]. Then, for each packet, *hash partition* between the m accumulator-counter pairs in order to decide on which one to use [1]. Since this choice must be consistent across the sender and receiver, both must use the same hash function in order for this to work [1]. When using such a bank of m accumulator-counter pairs, a single packet loss would only corrupt a single one of these pairs while the others remain usable [1]. The average delay of a subset of all sampled packets may then be determined by combining all usable accumulator-counter pairs [1].

However, using a bank only allows the LDA to handle low packet loss rates [1]. A high loss rate, combined with a high amount of throughput, would still be likely to quickly corrupt the entire bank [1]. To cope with this, instead of sampling every packet received, a fixed sampling probability p can be imposed on the bank with the goal of reducing the number of potentially *unusable* accumulator-counter pairs [1]. Since, in order to maintain consistency, a packet should be sampled at the receiver if and only if it was sampled at the sender, it is logical to use the same sampling rate at both ends [1]. A common hash function may then be used on each packet in order to apply this sampling rate [1]. Based on the rate of packet loss, a suitable sampling rate p, which maximizes the expected sample size, may then be determined [1].

In a realistic scenario however, the packet loss rate is usually unknown and prone to change over time [1]. This can be dealt with by using an array of n banks, each one tuned to a different packet loss rate through its sampling rate p_i [1]. When using multiple banks, it is advantageous for those banks to have *disjoint* sampling sets, meaning that no packet is ever sampled by more than one bank, since this opens up the possibility of combining all usable accumulator-counter pairs across all banks in order to calculate the average latency from the greatest possible sample, without the possibility of some packets being counted multiple times [1]. If the sampling probabilities $p_1, \ldots, p_n$ were to be powers of $\frac{1}{2}$, disjoint sampling sets could easily be achieved by hashing each packet to a bitstring, where each bit has an equal probability of being 0 or 1 [1]. Then, use the number of leading zeroes of the bitstring to determine which bank, if any, will sample the packet [1].

3.1.3. The complete Data Structure. The full LDA consists of an $m \times n$ matrix of accumulator-counter pairs. Each of the n colums represents a separate *bank*, holding its own disjoint set of samples [1].

The update procedure when sampling a packet, as proposed in [1], is:

1) Calculate a uniform hash of the packet.
2) Use the hash and the sampling probabilities $p_1, \ldots, p_n$ to decide whether the packet should be sampled and which bank to use in that case.
3) Use the hash to select an accumulator-counter pair of the chosen bank.
4) Add the timestamp at which the packet was received to the timestamp accumulator and increment the packet counter.

At the end of each measurement interval, the sender's LDA is transmitted to the receiver (or vice versa) and the average latency is estimated [1]. Let T_A and T_B be the sum of all usable timestamp accumulators of the sender's and receiver's LDAs respectively [1]. A timestamp accumulator-counter pair is considered usable if its packet counter at the receiver matches the corresponding one at the sender [1]. Let S be the effective sample size, which is the sum of all usable packet counters at the receiver (or the sender) [1]. Following from this, the average delay estimate D is

$$D = \frac{1}{S}\left(T_B - T_A\right) \tag{4}$$

as described in [1].

3.2. Lossy Difference Sketch

The Lossy Difference Sketch (LDS) is a probabilistic data structure proposed by Sanjuas et al. in [2]. It builds on the basic ideas behind the LDA (Section 3.1) and is meant to be used to estimate the average packet latency between a sender and a receiver, while being lightweight in terms of memory usage and computational overhead. Just as the LDA, it is used to conduct *one-way* measurements [2]. However, unlike the LDA, which estimates the average latency of *all* packets sent from a sender to a receiver, it produces *per-flow* estimates, meaning that a separate latency estimate can be obtained for each flow observed [2]. In this context, we consider a *flow* a tuple of the source and destination address, source and destination port and the protocol [2].

The motivation behind the LDS is based on the observation that packet latencies might differ considerably between flows, meaning that average latencies over all packets may not prove sufficient in order to detect and analyze application-specific network performance issues [3].

3.2.1. Basic Idea. Much like an LDA, the LDS estimates average latencies by aggregating timestamps and comparing them in periodic time intervals [2]. However, in order to conduct *per-flow* measurements, it needs to distinguish between flows. It does so in a probabilistic manner by *hash-partitioning* between timestamp accumulator-counter pairs based on the flow f [2]. Additionally, each timestamp accumulator-counter pair also stores a *flow digest* [2] in order to detect potential inaccuracies caused by packet reordering between the sender and receiver [5]. We refer to a tuple consisting of a timestamp aggregator, packet counter and flow digest as a *bucket* [2].

3.2.2. Functionality. The LDS consists of a $R \times C$ matrix of *buckets* [2]. Each packet is sampled once per row [2].

Sampling Procedure. When sampling a packet, for each row, an index within that row is determined based on a hash of the flow f and row index i [2]. Then, the index is offset by a number up to k, determined by hash of the full packet, resulting in an even distribution of samples across k adjacent buckets, k being an adjustable parameter of the LDS [2]. The bucket at the resulting index is then updated, meaning that the packet's timestamp is added to the accumulator, the packet count is incremented and the flow digest is XOR-ed with a hash of the full packet [2]. Inspired by the LDA, this index shift is done to reduce the impact of packet loss and, in this case, packet reordering by distributing samples of a flow across k different buckets [2]. We note that this allows for potential collisions between samples of flows [2] and will discuss how the LDS attempts to find the buckets with the least interference in order to accurately estimate an average delay. Further, it is evident that all hash functions used must be equivalent on the sender and receiver as to maintain consistency [2].

Estimating Average Delays. In order to estimate the average latency of a flow f, first select all usable buckets of the ones samples of f were collected in [2]. A bucket is considered usable if both its packet count and its flow

digest at the sender and receiver match [2]. Then, find the bucket with the lowest packet count n [2]. We are assuming this is the one with the least interference from other flows [2]. Then, from the previously determined set of usable and related buckets, select those that have sampled at most $n(1+\alpha)$ packets, where α is a configurable parameter, and calculate the average latency by combining them [2]. Reference [2] suggests using a value of 0.1 for α.

3.3. Reference Latency Interpolation

Reference Latency Interpolation (RLI), proposed by Lee et al. in [3], is an alternative technique to estimating *per-flow* packet latencies between a sender and a receiver. It leverages the observation that packets belonging to different flows tend to experience similar latencies when sent with little delay in between each other [3]. Just like the previously discussed approaches, RLI is used to conduct *one-way* latency measurements [3].

When using RLI, the sender periodically sends a reference packet holding a timestamp to the receiver and the receiver then uses the timestamp to calculate the reference packet's delay [3]. For every non-reference packet received, its delay is then estimated using linear interpolation between the previous and the next reference packet's delays as well as factoring in the individual packet's size and the link capacity [3]. Since a packet's delay can only be estimated after another reference packet was received, all information needed to estimate its delay, namely its flow identifier and timestamp of its arrival, are stored in a buffer until the delay can be estimated [3].

It should be mentioned that the decision of when to inject a reference packet is not straightforward. For example, injecting a reference packet every n packets sent could work well under high network load but might result in too few packets being injected under low load, lowering the accuracy of the estimate [3]. Injecting reference packets in fixed time intervals instead would solve the accuracy issues for low network load but in turn might result in too few packets being injected under high load [3]. Reference [3] therefore suggests a combination of both rules, effectively adapting the injection rate dynamically based on the network load.

4. Comparison

We now compare the discussed approaches in Table 1. The properties we are comparing them against are the type of measurement (overall or per-flow estimate), robustness against packet loss and reordering, network overhead and accuracy.

4.1. Robustness

While the LDA and LDS make use of *hash partitioning* in order to deal with packet loss, potentially losing a portion of their samples, RLI samples every packet that was not lost [1]–[3]. The LDA assumes packets arrive in the same order they were sent and has no protection against packet reordering [1], [5], while the LDS detects and avoids affected samples [2] and RLI is not affected by it since it does not aggregate timestamps [3].

TABLE 1: Comparison between LDA, LDS and RLI

Type	Measurement	Loss	Reord.	Overh.	Acc.
LDA	overall	+	–	++	++
LDS	per-flow	+	+	++	++
RLI	per-flow	++	++	+	+

Measurement: Overall or per-flow measurement
Loss: Robustness against packet loss
Reord.: Robustness against packet reordering
Overh.: Network overhead
Acc.: Accuracy of measurement/estimate

4.2. Network Overhead

Both the LDA and LDS do not incur any network overhead during their measurement intervals and transmit the data structure once at the end of each measurement interval [1], [2]. Since RLI regularly injects reference packets, which also affect the forwarding behavior of routers, we consider its overhead slightly larger [3].

4.3. Accuracy

Evaluations of the LDA and LDS in [1] and [2] respectively have shown that both suffer a very low mean relative error under low ($< 1\%$) packet loss. However, since different flows may interfere with each other within an LDS, it tends to be more accurate for larger flows than smaller ones because large flows carry more weight within their latency estimate [2]. When facing packet loss rates below 5%, the LDA suffers a reasonably small mean relative error of 3% to 9% [1]. RLI, on the other hand, experiences a mean relative error of 10 to 12% for moderate to high link utilization and around 30% for low link utilization [3]. A direct comparison between the LDS and RLI in [2] showed that the LDS outperforms RLI in terms of accuracy under low ($< 1\%$) packet loss.

5. Conclusion

In this paper, we have discussed three different probabilistic approaches to estimating packet latencies between two network points. First, we have discussed the basic idea behind aggregating timestamps and how the Lossy Difference Aggregator (Section 3.1) leverages it. We then proceeded with the Lossy Difference Sketch (Section 3.2), which makes use of the basic ideas behind the LDA, but estimates *per-flow* latencies [2]. Finally, we have looked at Reference Latency Interpolation (Section 3.3), an alternative approach to estimating *per-flow* latencies by interpolating between latencies of reference packets [3]. Following that, we have compared the three approaches in Section 4, where we have seen that both the LDA and LDS usually provide more accurate latency estimates than RLI, though they are less robust against packet loss and reordering.

References

[1] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese, "Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 255–266, 2009.

[2] J. Sanjuàs-Cuxart, P. Barlet-Ros, N. Duffield, and R. R. Kompella, "Sketching the delay: tracking temporally uncorrelated flow-level latencies," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 483–498.

[3] M. Lee, N. Duffield, and R. R. Kompella, "Not all microseconds are equal: Fine-grained per-flow measurements with reference latency interpolation," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 27–38.

[4] J. C. Eidson, M. Fischer, and J. White, "Ieee-1588™ standard for a precision clock synchronization protocol for networked measurement and control systems," in *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*, 2002, pp. 243–254.

[5] M. Lee, S. Goldberg, R. R. Kompella, and G. Varghese, "Fine-grained latency and loss measurements in the presence of reordering," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 2011, pp. 329–340.

Industrial Ethernet: Challenges and Advantages

Moritz Werner, Florian Wiedner*, Christoph Schwarzenberg*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: moritz.werner@tum.de, wiedner@net.in.tum.de, schwarzenberg@net.in.tum.de

***Abstract*—The IEEE 802.3 Ethernet standard was published in 1985 and quickly became the predominant standard of digital traffic in small, local networks all the way up to the internet itself. Paired with the TCP/UDP IP stack, it is responsible for nearly all internet traffic, including website-requests, file transfers and communication in general. Requirements for industrial applications are vastly different due to physical constraints and software necessities. The goal of this paper is to describe and categorize some variants of industrial ethernet protocols (IE), compare their efficiency, and evaluate their respective advantages and disadvantages. Due to its frame structure and protocol modalities, EtherCAT can reach lower response times than POWERLINK, Sercos III and PROFINET under normal circumstances.**

***Index Terms*—protocols, ethernet, process control, profinet, ethercat, sercos III, powerlink, industrial ethernet**

1. Introduction

Producing goods and tools is an inherently human trait. Utilizing steam power in factories was a key milestone in human evolution, often called industry 1.0. The second generation switched to electrical systems and machines, industry 3.0 saw the use of simple programmable controls and computers. We are currently implementing industry 4.0, combining information and production, physical and digital entities into a complex interconnected environment. Incorporating the IEEE 802.3 Ethernet in the majority of our information exchange was an important step of standardization, but the abilities it provides are not viable for use in industrial settings. The most important requests from manufacturers were "realtime guarantees, such as a maximum transfer time, a jitter [small fluctuation in the transmission time accuracy] not exceeding some threshold, or some guaranteed bandwidth" [1], conditions the aforementioned standard cannot provide. In turn, many companies had their own proprietary network solutions, not suited for interoperability and scalability [2]. This changed with the introduction of a number of industrial ethernet protocols, which will be discussed in section 5. They are based on the principles of a 'master' node, which can be an ordinary computer providing data and telemetry for surveillance. This master is responsible for the actuation of several 'slave' nodes, they are themselves control single machines, receive instructions from the master, execute them and possibly send information back to the master. The time needed to send data from the master to all connected slaves and receive their answers is called cycle time. Based on its protocol structure, EtherCAT can provide shorter cycle times than POWERLINK, Sercos III and PROFINET, which is shown in Section 6.

2. Physical Conditions in Industrial Environments

Native ethernet works best in well-controlled environments like offices or data centers. Industrial environments like factories or power plants have substantially different working conditions depending on manufacturing steps, used resources, possible chemicals and desired products, which affect networking if not sufficiently accounted for. Controller, machines, sensors and the cabling itself might be subjected to hazardous environments, high temperatures, dust particles, electromagnetic interference or vibrations [3], [4]. The induced loss of packets or wrong transmissions have to be solved in order to make ethernet viable in such conditions. Solutions are already available in the form of distinctive electronics and ethernet cables with IPX-rating for waterproofing, special isolation and Twisted Shielded Pair configurations against noise and interference, as well as industrial connectors between machine and cable.

3. Requirements of Industrial Ethernet

To implement an industrial ethernet protocol, it has to fulfill certain requirements dictated by the equipment on site. Cycle times should be as short as possible to improve reaction times of important components like motors, valves or robotic arms. Fluctuations in data transmission speed (jitter) should also be minimized to allow for more consistent cycle phases, which is very important for polling-style industrial ethernet like POWERLINK or PROFINET as they rely on time scheduling for sending their datagrams [5]. As connecting networks from a simple control level to the factory floor, enterprise level, and possibly all the way up to the internet is one of the goals of industrial ethernet, network security is of high concern to ensure a safe and reliable production environment.

4. Industrial Ethernet - State of the Art

A study on the usage of industrial ethernet done in 2022 revealed that PROFINET and Ethernet/IP are the most used protocols with both around 14%, and EtherCAT at around 11%. IE protocols as a whole are used by 66% of participants, field bus technologies like DeviceNet or

 doi: 10.2313/NET-2024-04-1_06

PROFIBUS made up around 27% market share. Interestingly, 7% reported wireless networking as their main mode of networking [6].

5. Implementations

The following sections describe the functionality of four different industrial ethernet implementations, their frame/packet structure and network layouts.

5.1. Ethernet

Ethernet conforming to the IEEE 802.3 standard is the basis of commercial, office-related and international digital traffic. It allows for a uniform way of TCP/IP connections, file and mail transfer as well as some implementations of industrial ethernet, i.e., PROFINET, EtherCAT or Ethernet/IP.

The ethernet frame as shown in Figure 1 consists of a few fields [1]: starting with a 7 Byte preamble, followed by a 1B start of frame field. After that, the destination and source MACs are addressed with 6B, respectively. The Ether-Type is transmitted in a 2B field if its value is above or equal to 1536, or the total frame length if it is below or equal to 1500. The next one houses between 46B and 1500B of user-data, which can, for example, be an IP-, ICMP-, or an IE-packet. It is completed by the 4B long FCS field, which is used to verify the frames correct transmission.

5.2. PROFINET

Mainly defined by Siemens and supported by PROFIBUS International, the PROFINET protocol suite is a group of follow-up protocols to PROFIBUS [2], [5], [8]. There are currently four classes of PROFINET named CC-A through CC-D, providing different levels of network traffic. Class A is for cyclic and acyclic data transfer, class B builds upon class A and allows for reduced cycle times and real-time traffic. Following up, class C enables isochronous traffic while shortening cycle times even further, and the latest version (CC-D) released in 2019 incorporates time-sensitive-networking [4], [9].

PROFINET uses singular frames addressed to individual devices in the network, a procedure called individual frame (IF) [10]. Further developments make use of fast forwarding, dynamic frame packing and fragmentation of TCP/IP telegrams to increase throughput and decrease cycle times down to 31.25μs [5]. However, this cycle time is not achievable in real world applications, as transmission errors and switches introduce time delays. To mitigate time losses and decrease network jitter, special cut-through switches are needed, which unlike normal managed network switches do not operate in the store-and-forward mode [1], [8]. Also, both the master and slave devices need particular hardware in the form of ASICs to handle the fast processing of data [8].

The PROFINET telegram is made of a few key elements, such as a 2B Frame-ID, the payload itself and status control-fields, as seen in Figure 2a.

A typical PROFINET cycle has two distinct phases: first, cyclic real-time data is sent from the master to each slave device, awaiting the respective answer, followed by acyclic real-time data used for alarms and non-real-time data like TCP/IP. Should a slave miss its allocated time slot for sending data to the master, the information is simply discarded and the device can try to do so in the next cycle [5], [8]. PROFINET networks can have a number of topologies, e.g. line, ring or star [8], and can "Operate properly and keep temporal guarantees" in the presence of 802.3 compliant node [1]. According to a paper by Prytz [8] and [5], PROFINET can achieve a network jitter of 1μs and a minimum cycle time of 31.25μs.

This paper will focus on the PROFINET IRT (CC-C) variant in the evaluation section.

5.3. POWERLINK

Similar to PROFINET, POWERLINK uses individual frames and a cycle consisting of cyclic and acyclic phases, but with the added benefit of being completely software-based. The master/slave designation for network devices is given in the form of managing nodes (MN) and controlled nodes (CN) [11], [12].

A POWERLINK telegram contains fields for the message type, 8bit each for target and source node, plus a 61B to 1497B payload. Figure 2b shows additional space reserved at the beginning for the future.

Each cycle starts with the start of cycle (SoC) message sent via broadcast by the MN to ensure synchronization of all CNs. In contrast to PROFINET, the MN polls every CN individually by sending a poll request (PReq) packet and thus allowing the corresponding slave node to send a poll response (PRes) to all nodes. Again, PRes packets which are not received in a given timeframe are disregarded and the MN moves on to the next CN. To mark the end of this phase, a start of acyclic (SoA) message is broadcast and CNs are polled for non-real-time traffic. Finally, the cycle goes through an idle period and starts all over again. A paper by Cena et al. [13] simulated the cycle time to be 1.3ms, while B&R Industrial Automation GmbH [11] lists it around 100μs.

In principle, POWERLINK supports any topology like star, ring or daisy-chain for up to 240 devices per network [11].

5.4. EtherCAT

Contrary to PROFINET and POWERLINK, the EtherCAT master does not send an individual frame to each slave, instead a single frame with positional fields directed at each slave is sent per cycle. The frame passes through one node after the other and updates its content on the fly until the last slave is reached, which sends the frame back to the master using the full-duplex mode of ethernet [14]. This method is called Summation Frame (SF) and allows for a very basic implementation of the master node, which can be fully realized in software requiring only standard ethernet hardware [8], [10]. Slave nodes, however, need to be able to quickly process the datagram and possibly change their allocated space along with calculating a new FCS, therefore they rely on an EtherCAT slave controller with the necessary hardware capabilities. Up to 65535 devices can be connected to a single EtherCAT segment [14].

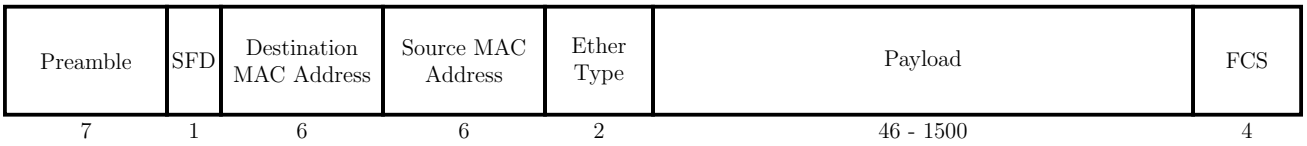
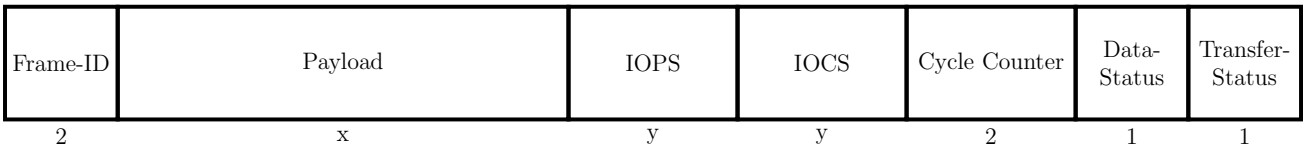
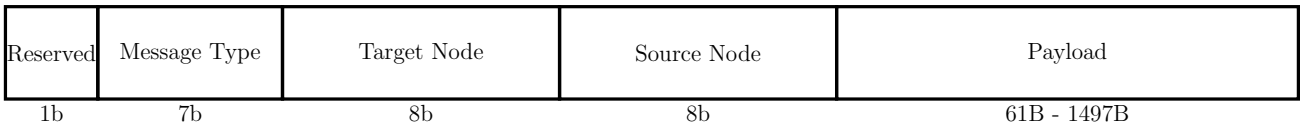


Figure 1: IEEE 802.3 Ethernet frame fields and their respective sizes in Byte [7]



(a) PROFINET, field sizes in Byte [10]



(b) POWERLINK, header fields in bit, data in Byte [11]

Figure 2: A PROFINET IRT frame, and a POWERLINK frame

The frame itself consists of a 2B EtherCAT header and a number of telegrams addressed to the slaves. Each telegram starts with a 10B telegram-header, followed by the actual payload and a 2B working counter as displayed in Figure 3. This counter is used for telemetry and diagnostics, but it can also be deactivated to increase packet space for user transmissions [10].

There is also the EtherCAT P variant to supply slave nodes with power and data using the same ethernet cable. Just like POWERLINK, many network topologies are feasible with EtherCAT [14]. At the Hannover Messe in 2012, the EtherCAT Technology Group demonstrated an EtherCAT network with a cycle time of 12.5μs, more than twice as fast as PROFINET [14].

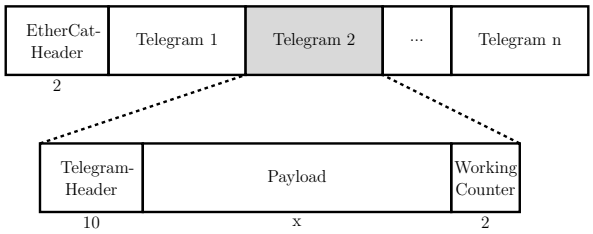


Figure 3: EtherCAT frame fields, sizes in Byte [10]

5.5. Sercos III

Sercos III is similar to EtherCAT in the way that it also uses a summation frame method, however the path on which frames are sent is not limited to a single direction. Rather, when a slave device is finished with modifying the frame of the current cycle, it can propagate the changes to multiple neighboring devices [1].

"Sercos supports direct cross communication, which enables real-time data exchange between any Sercos devices within one communication cycle" [15].

The real-time data field houses the main master to slave data transfer as well as any other possible device-device traffic, independent of the nodes' role. Slave to slave communication has its own fields in the frame, named CC channel connection 1 to n as seen in Figure 4. Others are used for hot-plugging new devices, meaning connecting additional nodes without a network restart. Cycle times can be as low as 31.25μs, just like PROFINET [1], [15].

In a line configuration, the master sends its telegram sequentially to the daisy-chained slaves, and the last device loops it back allowing all devices to see changes made by all others. A ring topology ensures additional protection against downtime by redundant cabling, should errors occur [15]. With nodes using standard 802.3 Ethernet in the same network, temporal guaranties for telegram transmission are lost [1].

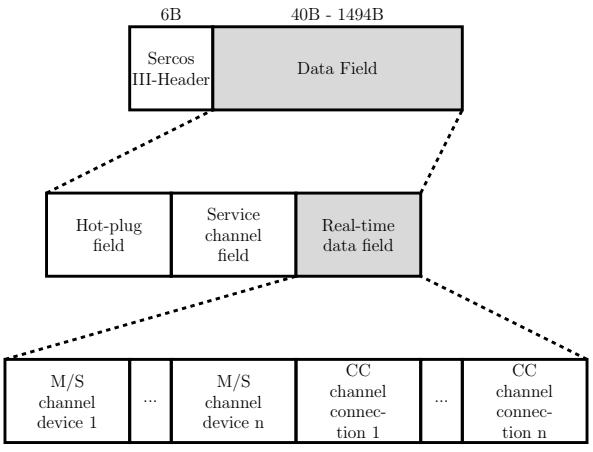


Figure 4: A Sercos III frame with its data fields [15]

6. Evaluation

As PROFINET IRT and EtherCAT are currently among the most prevalent IE protocols, most research papers are only comparing these two. Therefore, this section will rely on data from the respective developer

 doi: 10.2313/NET-2024-04-1_06

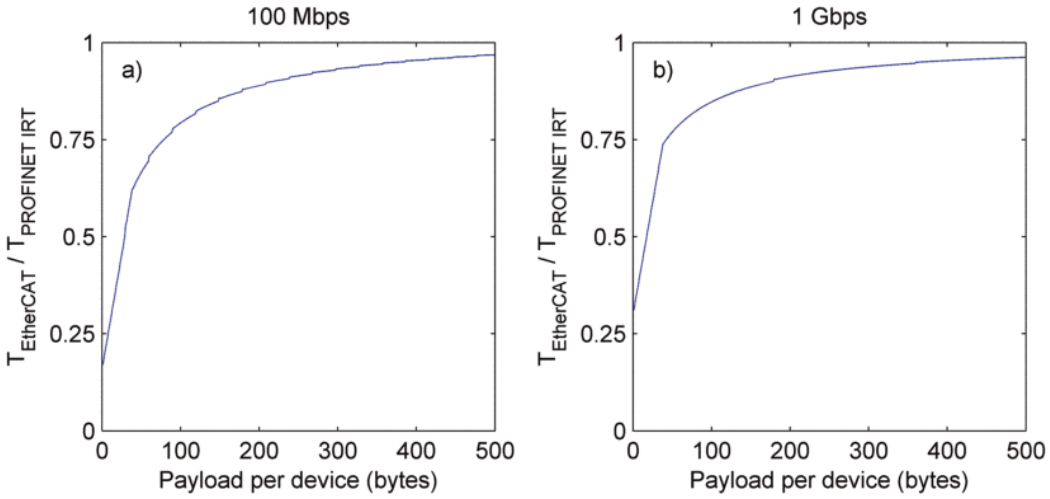


Figure 5: Relative performance of EtherCAT and PROFINET in a line topolgy network with 50 devices [8]

of POWERLINK and Sercos III, with the request not to consider the data as scientifically proven. The open-source POWERLINK manual "Communication Profile Specification" [11] names a lower bound of 100μs as its cycle time. Sercos III is specified to have a cycle time between 31.25μs and 1000μs, depending on the slave device number, the datagram size per slave, and whether the cross-device communication is enabled [15].

The following diagrams and data were produced in papers by Prytz [8] and by Wu and Xie [13]. Prytz presented a total of six experiment configurations with payload sizes of 16, 32 and 100B, combined with bandwidths of 100 MBps and 1 Gbps. The paper by Wu and Xie tested "an industrial NCS [networked control system] with 5 controlled plants [...] deployed with 5 sensors and 5 actuators connected onto the network backbone in a line topology" [13]. Their proceedings found that using a line topology, cycle times are shorter for EtherCAT for each payload size and both bandwidths [8]. The conclusion for relative performance of EtherCAT and PROFINET in a line topology is exemplary shown in Figure 5.

The authors have already justified their findings [8], [13]: EtherCAT being an SF protocol means a single frame containing all information is sent per cycle; PROFINET has to send an individual ethernet frame to every node, creating a lot of overhead and subsequently lowering performance. Especially in use cases with small payloads and a low number of slave devices, EtherCAT has substantially better performance than PROFINET, as that configuration is the most optimal for EtherCAT while simultaneously being suboptimal for PROFINET. When accounting for transmission errors, however, IF protocols could achieve a higher performance compared to SF protocols, depending on the error rate. As errors in the transmission only affect single devices, the impact of erroneous transmission is contained to a single cycle of a single device, instead of all connected slaves.

7. Conclusion

This paper briefly explained four different industrial ethernet protocols, compared their approaches to provide realtime control of various devices and evaluated the performance of EtherCAT and PROFINET in a number of situations.

Of course, developers of protocols want to present their implementations in the best possible way, but their performances are often more theoretical than actually achievable. Although cycle times as low as 31.25μs are impressive, in the case of Sercos III they were achieved in a network made of only seven devices, not using the slave-slave communication which made that protocol unique, with the optimal line topology and without the presence of any transmission errors [15]. In the author's opinion, these conditions are more akin to a lab environment than an industrial one with physical challenges, as described in section 2.

Currently, most devices can run successfully on only a few bytes of process information per cycle, in the future more data might have to be shared with individual nodes in ever larger networks, which would make SF protocols less viable. Advances in faster and more specialized hardware for IE protocols as well as more streamlined software will shape the future development of industrial applications. Although ethernet jumbo frames with up to 9000B of payload space exist, that might just be not enough one day. The need to provide both workplace safety for the personnel on the factory floor and digital security for appliances arises as connecting previously separated industrial networks to the corporate structure becomes more common; some protocols already incorporate such measures, and a proposal was made by Giehl and Plaga in their paper [16]. Industrial ethernet forms the backbone of today's production systems and will most certainly be used and improved upon in the future.

References

[1] J.-d. Decotignie, "The many faces of industrial ethernet [past and present]," *IEEE Industrial Electronics Magazine*, vol. 3, no. 1, pp. 8–19, 2009.

[2] M. Felser and T. Sauter, "Standardization of industrial ethernet - the next battlefield?" in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, 2004, pp. 413–420.

[3] J. A. Kay, R. A. Entzminger, and D. C. Mazur, "Industrial ethernet-overview and best practices," in *Conference Record of 2014 Annual Pulp and Paper Industry Technical Conference*, 2014, pp. 18–27.

[4] Z. Lin and S. Pearson, “An inside look at industrial ethernet communication protocols,” https://www.ti.com/lit/wp/spry254b/spry254b.pdf, 2018, [Online; accessed 30-August-2023].

[5] P. N. e.V., “Profinet - the leading industrial ethernet standard,” https://www.profibus.com/technology/profinet, [Online; accessed 22-September-2023].

[6] T. Carlsson, “Industrial networks keep growing despite challenging times,” https://www.hms-networks.com/news-and-insights/news-from-hms/2022/05/02/industrial-networks-keep-growing-despite-challenging-times, [Online; accessed 30-September-2023].

[7] “Ieee standard for ethernet,” *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pp. 1–7025, 2022.

[8] G. Prytz, “A performance analysis of ethercat and profinet irt,” in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 408–415.

[9] I.-S. GmbH, “Profinet conformance class,” https://www.indu-sol.com/support/glossar/conformance-class/, [Online; accessed 25-September-2023].

[10] J. Jasperneite, M. Schumacher, and K. Weber, “Limits of increasing the performance of industrial ethernet protocols,” in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, 2007, pp. 17–24.

[11] B. I. A. GmbH, “Powerlink,” https://www.br-automation.com/en/technologies/powerlink/, [Online; accessed 26-September-2023].

[12] G. Cena, L. Seno, A. Valenzano, and S. Vitturi, “Performance analysis of ethernet powerlink networks for distributed control and automation systems,” *Computer Standards & Interfaces*, vol. 31, no. 3, pp. 566–572, 2009, industrial Networking Standards for Real-time Automation and Control. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548908000433

[13] X. Wu and L. Xie, “Performance evaluation of industrial ethernet protocols for networked control application,” *Control Engineering Practice*, vol. 84, pp. 208–217, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066118303691

[14] E. T. Group, “Ethercat - the ethernet fieldbus,” https://ethercat.org/en/technology.html, [Online; accessed 24-September-2023].

[15] S. I. e.V., “Sercos technology: Proven, easy, fast, open,” https://www.sercos.org, [Online; accessed 28-September-2023].

[16] A. Giehl and S. Plaga, “Implementing a performant security control for industrial ethernet,” in *2018 International Conference on Signal Processing and Information Security (ICSPIS)*, 2018, pp. 1–4.

Predictive Modelling for Next API Call Sequence in Content Delivery Networks

Galiiabanu Bakirova, Markus Sosnowski *
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: galiiabanu.bakirova@tum.de, sosnowski@net.in.tum.de

***Abstract*—Content Delivery Networks (CDNs) play a crucial role in enhancing user experience by caching frequently accessed content. However, challenges arise when users request uncached content, leading to potential disruptions and impacts on website performance. To address this, researchers advocate for proactive caching, pre-emptively storing predicted future requests. This paper explores the application of a language model to server logs for predicting users' API requests. Training the model on logged API calls, we analyze the potential of language models in gaining insights into user behavior. Despite challenges with dynamic data, the model detects recurring patterns and learns API semantics. While results may vary for sites with dynamic structures, this approach opens avenues for future improvements, such as introducing probability thresholds or focusing on specific API endpoints. Challenges persist, requiring each website to train its own model based on its API structure. Our exploration provides valuable insights into the possibilities and limitations of language model-driven API request prediction.**

***Index Terms*—content delivery networks, api, prediction**

1. Introduction

In our rapidly evolving world, website owners often rely on Content Delivery Networks (CDNs) to speed up loading times for end-users. Yet, this decision poses its own set of challenges, including determining which content to cache on the CDN and minimizing cache misses [1].

CDNs typically store popular and frequently accessed content, a strategy that significantly reduces latency and improves user experience. However, challenges arise when a user requests content that is not readily available in the CDN's cache. In such cases, the CDN must either retrieve the resource from the origin server or return an error message to the user. This can lead to disruptions in the user experience and potentially impact overall website performance [2].

To address this challenge, researchers have proposed the idea of predicting users' next requests and preemptively caching the required resources. In their survey, Nanopoulos et al. [3] advocated for storing such items in the cache prior to an explicit request being made. This approach, known as proactive caching, aims to minimize the latency associated with fetching resources from origin servers, ensuring that users enjoy seamless and responsive online experiences.

Building upon this concept, machine learning models have emerged as promising tools for predicting users' next requests. These models can analyze vast amounts of server logs and historical data to identify patterns and predict user behavior. This information can then be used to preload relevant resources in the CDN cache, further enhancing the user experience and reducing latency. [4]

In this paper, we explore the application of a language model approach to server logs datasets to gain valuable insights into user behavior and predict their next API requests.

2. Background and Related Work

A Content Delivery Networks (CDN) is a network of servers strategically located around the world that help deliver web content to users with improved performance and reduced latency [5]. Websites and online platforms often utilize CDNs to enhance their services and meet the growing demands of users.

CDNs work by storing cached copies of web content, such as images, videos, and static files, on servers distributed across various geographic locations. When a user requests content, the CDN automatically directs the request to the server closest to the user, minimizing the distance the data needs to travel. This proximity helps reduce latency and improves the overall loading speed of web pages [2].

In addition to improving performance, CDNs can also help mitigate traffic spikes and distribute the load on servers, ensuring smooth and uninterrupted access to web content even during periods of high demand. By offloading the delivery of content to a CDN, websites can optimize their infrastructure, enhance user experience, and better handle global traffic [2].

2.1. Web Usage Mining

Colley et al. [6] were some of the first ones interested in predicting user's requests. Knowing user's intentions can create a seamless user experience, increase conversion and sales [7] if the users are recommended something they did not know they needed [8]. In the context of server and CDNs, knowing user intentions can be used to predict the user's next API request and preload the resources. This would decrease loading time for the users and ensure effective user interaction. [3]

There are several proposed web usage mining approaches for working on server logs [9]:

 doi: 10.2313/NET-2024-04-1_07

- Association rules - the technique for finding the web pages visited together. One disadvantage of the associate rules approach is that it does not take into account the notion of time difference.
- Frequent Sequences - considering ordered time-sensitive sequences. This technique tries to discover sequence patterns followed by users.
- Frequent Generalized Sequences - a relaxation of frequent sequences, that allows to study user's navigation in a flexible way. [10]

Gery et al. [9] evaluated the three web usage mining approaches on datasets of different sizes and discovered that the Frequent Sequences (FS) performed best in terms of accuracy. The authors emphasize the suitability of the FS technique for analyzing time series and propose an optimal user session time of 25 minutes.

Nigam et al. [11] tried a different approach to predicting user requests. The research studies the effect of Markov model depth on the user's next request prediction. Nigam et al. compare first-, second- and third-order Markov models for predicting the next web page. They propose metrics such as model generation time, prediction time prediction accuracy and coverage for measuring prediction success. In [11] Nigam et al. perform experiments on three datasets. However, the proposed test datasets only have between 29 and 92 different pages, which would correspond to the number of unique API calls.

2.2. Content Delivery Networks Scaling

Content networking is gaining popularity as a go-to technology because it significantly boosts enterprise network performance for media-rich content, all while keeping costs lower compared to traditional methods of web scaling [12]. As CDNs expand their user base, the content stored on CDNs becomes very diversified. Each content category imposes distinct demands on the CDN's caching systems. CDN has to introduce various configuration parameters in order to be able to serve such a wide range of content [13], [14]. Manual tuning of these parameters can be challenging.

With the pursuit of increased efficiency and reduction of cache misses, some reinforcement learning techniques to autonomously manage resources were proposed [13], [15], [16]. Current approaches to caching predominantly utilize "model-free" reinforcement learning (RL), where the system embarks on the learning process without knowledge of the underlying structure, free from any preconceived notions or biases about the task at hand [17], [18]. These systems learn decision-making through first-hand experience, guided by a reward mechanism that reinforces the right decision-making and encourages continued exploration of effective strategies.

While model-free RL holds immense promise for optimizing caching strategies, the RL community has identified three major hurdles that need to be addressed:

- Data-Intensive Learning: Model-free RL algorithms typically require vast amounts of training data, usually in the millions of samples. Accumulating such a large dataset can be time-consuming and resource-intensive [16].
- Overfitting Vulnerability: Model-free RL algorithms are susceptible to the risk of overfitting to the train data. This means that they may perform well on the training set but struggle to generalize to new and unseen data. In the context of caching, overfitting could lead to suboptimal caching decisions [19].
- Complex Debugging and Maintenance: Model-free RL algorithms can be sensitive to hyperparameters, which make them extremely difficult to debug [15], [20].

These challenges pose significant obstacles to the practical implementation of model-free RL for caching in CDN servers.

2.3. RNNs for API Requests Analysis

Reddy and Rudra [21] applied RNN for detecting injections in API requests. They compare three popular RNN approaches for sequential data analysis: bidirectional Vanilla-RNNs, bidirectional LSTMs and bidirectional gated recurrent units (GRU) for requests classification. The obtained results prove the effectiveness of RNN approach on a API request data. Reddy and Rudra were able to achieve the accuracy of 97% for the bidirectional LSTMs and 98,5% for bidirectional GRUs. Arivukarasi and Antonidoss [22] were also able to exploit natural language processing (NLP) approach with RNNs to detect phishing URLs. They achieved the highest accuracy of 98% using RNNs with LSTM layer.

3. Methodology

We applied the language model approach for training the API request prediction model. In order to train the language model for predicting the next API call, we used a recurrent neural network (RNN). RNNs are designed to work with sequential data and are, therefore, the perfect choice for processing a series of API requests. A model was created and trained with the TensorFlow Keras API. The architecture consists of three layers: Embedding layer, Long Short-Term Memory (LSTM) layer, and dense layer. The Long Short-Term Memory layer is the key to capturing long-term dependencies and is suitable for predicting the next API calls based on several previous calls. The architecture of the RNN used is depicted in Figure 1.

The embedding layer is responsible for converting the integer-encoded token (in our case, an API request) into a dense vector. This step is needed to detect semantic dependencies between API requests and meaningfully model them in a vector space.

The second layer, LSTM, is responsible for preserving the cell state and the hidden state of the machine learning model. [23] It operates on a read-write-forget principle. The network learns which information is relevant and will be needed later and which information can be forgotten. The main advantage of the LSTM layer in contrast to classical vanilla RNN is that it solves the vanishing/exploding gradient problem, which appears when passing the gradient recursively for n steps [24].

The third dense layer outputs probabilities over the vocabulary, in our case, the whole API request set, for a

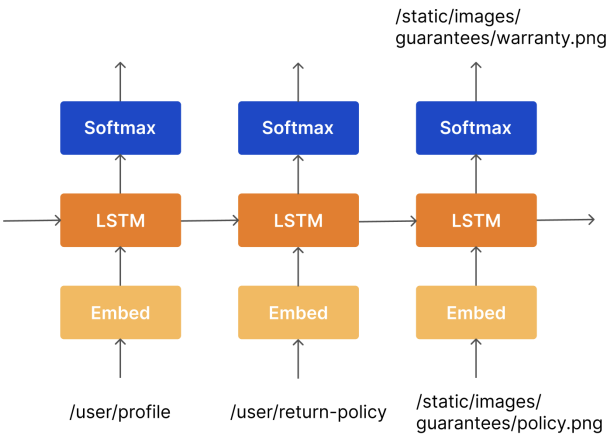


Figure 1: Architecture of RNN

given number of previous API calls. To achieve probabilistic prediction, the softmax activation function was used. For the training configuration, we applied a commonly used categorical cross-entropy as a loss function, adaptive optimizer Adam, and accuracy as a training metric. We used early stop approach to avoid overfitting, this method stops the training process when the accuracy on the validation set starts plateauing. The number of epochs therefore was different, between 5 and 20 epochs, dependent on the window size.

3.1. Data Preprocessing

Due to safety and commercial data considerations, there are limited real data server log datasets available. Many research groups opt to use server logs from their own intranet or record API traffic, creating a server logs dataset specifically for analytical purposes.

In order to test the proposition of being able to predict the next user request, we used an open dataset of server logs of an Iranian online shop, "Online Shopping Store - Web Server Logs" [25]. The dataset comprises more than 10 million logs in Common Log Format (CLF) that Apache uses and contains some valuable information about website usage [26].

Standard CLF server log looks like this:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326
"http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
```

It contains the client's IP address, request time, method used by the client (GET), information on requested resources (/apache_ pb.gif) and the protocol used (HTTP/1.0). The log also contains the respond status code (200), size of the object returned to the client (2326) as well as the referrer and user agent.

For the purposes of demonstrating the idea of predicting the next user request, users are distinguished by a client IP address, and the sessions are limited to a 20-minute time frame. The sliding window technique was applied to create training and test sets. After testing several sizes of the window, the best results were achieved with window size 4, where the fifth request should be predictable from the previous four. The model was also trained using window sizes ranging from 2 to 6, but the larger window sizes led to overfitting and performed worse on a test dataset.

TABLE 1: Dataset Statistics

Number entries	Number unique API requests	Number unique clients	Number identified 20-minute sessions
10 364 865	893 045	258 445	352 296

3.2. Tokenization

Tokenization is the process of converting words into integers for further model training. We used a custom tokenizer in order to be able to treat the whole API request as a single word. The custom tokenizer handles special characters such as underscores and slashes within the request to create a meaningful vocabulary. Sequences are then padded to ensure uniform length of the vectors.

4. Results

In this section, we discuss the initial data and the obtained results.

In the Table 1 you will find statistical metrics of the dataset used.

We tested several window sizes to determine the number of previous requests on which the prediction for the next API request will be based, ranging from 2 to 6. Figure 2 shows the accuracy on training and test sets with regard to window size. The results state that bigger window sizes, such as 5 or 6, clearly lead to overfitting since the accuracy difference on both sets becomes more discrepant. The peak of accuracy on the test set appears when applying the window size, capturing 3 or 4 requests. The exact accuracy values can be found in Table 2.

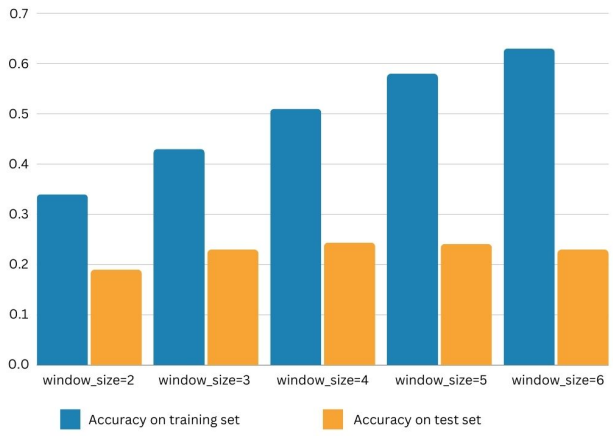


Figure 2: Accuracy on training and test set for different window size parameter

We only focused on the accuracy metric for the API request prediction. The resulting model could not overcome the 24% accuracy on a test set. This might seem low; however, it is worth mentioning that the website,

TABLE 2: Accuracy according to window size

window size	Accuracy on training set	Accuracy on test set
2	34.27%	19.07%
3	43.05%	23.04%
4	53.16%	24.39%
5	58.33%	24.11%
6	62.86%	22.98%

TABLE 3: Prediction Examples

Prediction	Actual
/image/57378/ productModel/100x100	/image/57124/ productModel/100x100
/static/images/ guarantees/warranty.png	/static/images/ guarantees/warranty.png
/static/images/amp/blog.png	/static/images/amp/blog.png

where the API traffic was recorded, is an online shop, containing large amount of dynamic data. The API requests themselves often contain product id, product images paths and other fine-granular details. Here are some examples of such requests:

```
/product/30910
/image/15474?name=1387476275_tc16.jpg&wh=200x200
```

This is, of course, very hard to predict and may be even impossible considering the continuously changing online shop assortment. In the training set, the size of the vocabulary or the token list, in our case the number of distinct API requests, was about 6000 for the training set with 300000 requests. The vocabulary size was dependent on the current batch.

Often, the model would predict the right API endpoint but fail to guess specific resource ID of the product or image. You may find an example of this in the first row of the Table 3.

However, the model shows better performance for the identified patterns, and dynamic data unrelated APIs such as return policy or guarantee resources. (See second and third examples in the Table 3.)

We also tested a hypothesis of building in a threshold on probability predictions, which would only preload data if the probability was sufficient. Unfortunately, this strategy failed to satisfy expectations. Instead, the most often correctly predicted requests were simply the most frequent ones.

Another suggestion on how to utilize the obtained tool could be to only restrict the next API request predictions for certain API endpoints. The most prominent example would be search endpoint. For the user's next search prediction, the larger window size would also be applicable [27].

5. Conclusion and Future Work

In pursuit of our goal to apply a language model approach to API request prediction, we trained a language model on a provided set of logged API calls. The vocabulary of the language model consists of all unique recorded API requests, and the word sequences are modeled based on user sessions.

Due to the large number of possible API URLs and dynamic data such as product IDs or image source ID, our effort did not yield impressive results. However, it still provided valuable insights into the future possibilities and limitations of the API request prediction. The model was able to detect some reoccurring patterns and learn the semantics of the API endpoint. This approach could be used for websites with a static structure and a limited number of API endpoints.

Some possible solutions to the low accuracy problem could be limiting the model to a certain set of API dynamic data-insensitive endpoints. The main drawback remains the fact that each website would have to train its own machine learning model based on the API structure.

In the future, the idea of successively guessing the next resource endpoint of the API URL should be examined. In this approach, the API paths would not be treated as single tokens but could be split up into hierarchical resource endpoints. It could exploit the hierarchical URL path structure.

References

[1] D. S. Berger, "Towards lightweight and robust machine learning for cdn caching," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, ser. HotNets '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 134–140. [Online]. Available: https://doi.org/10.1145/3286062.3286082

[2] E. Ghabashneh and S. Rao, "Exploring the interplay between cdn caching and video streaming performance," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 516–525.

[3] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "Effective Prediction of Web-user Accesses: a Data Mining Approach," Aug. 2001.

[4] D. S. Berger, "Towards Lightweight and Robust Machine Learning for CDN Caching," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. Redmond WA USA: ACM, Nov. 2018, pp. 134–140. [Online]. Available: https://dl.acm.org/doi/10.1145/3286062.3286082

[5] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[6] R. Cooley, B. Mobasher, and J. Srivastava, "Web mining: information and pattern discovery on the World Wide Web," in *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*. Newport Beach, CA, USA: IEEE Comput. Soc, 1997, pp. 558–567. [Online]. Available: http://ieeexplore.ieee.org/document/632303/

[7] M. A. T. Pratama and A. T. Cahyadi, "Effect of user interface and user experience on application sales," *IOP Conference Series: Materials Science and Engineering*, vol. 879, no. 1, p. 012133, jul 2020. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/879/1/012133

[8] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 158–166.

[9] M. Géry and H. Haddad, "Evaluation of web usage mining approaches for user's next request prediction," in *Proceedings of the 5th ACM international workshop on Web information and data management*. New Orleans Louisiana USA: ACM, Nov. 2003, pp. 74–81. [Online]. Available: https://dl.acm.org/doi/10.1145/956699.956716

[10] W. Gaul and L. Schmidt-Thieme, "Mining Web Navigation Path Fragments," Aug. 2000.

[11] B. Nigam, S. Tokekar, and S. Jain, "Evaluation of Models for Predicting User's Next Request in Web Usage Mining," *International Journal on Cybernetics & Informatics*, vol. 4, no. 1, pp. 01–12, Feb. 2015. [Online]. Available: http://www.airccse.org/journal/ijci/papers/4115ijci01.pdf

[12] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing cdn-isp collaboration to the limit," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, p. 34–44, jul 2013. [Online]. Available: https://doi.org/10.1145/2500098.2500103

[13] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[14] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "{AdaptSize}: Orchestrating the Hot Object Memory Cache in a Content Delivery Network," 2017, pp. 483–498. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/berger

[15] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," Jan. 2019, arXiv:1709.06560 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1709.06560

[16] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," Oct. 2017, arXiv:1710.02298 [cs]. [Online]. Available: http://arxiv.org/abs/1710.02298

[17] M. Lecuyer, J. Lockerman, L. Nelson, S. Sen, A. Sharma, and A. Slivkins, "Harvesting Randomness to Optimize Distributed Systems," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. Palo Alto CA USA: ACM, Nov. 2017, pp. 178–184. [Online]. Available: https://dl.acm.org/doi/10.1145/3152434.3152435

[18] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*. Barcelona, Spain: IEEE, Aug. 2014, pp. 917–921. [Online]. Available: http://ieeexplore.ieee.org/document/6933484/

[19] "Faulty reward functions in the wild." [Online]. Available: https://openai.com/research/faulty-reward-functions

[20] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[21] S. R. A and B. Rudra, "Evaluation of recurrent neural networks for detecting injections in api requests," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0936–0941.

[22] M. Arivukarasi and A. Antonidoss, "Performance analysis of malicious url detection by using rnn and lstm," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020, pp. 454–458.

[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[24] ——, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[25] F. Zaker, "Online Shopping Store - Web Server Logs," May 2021. [Online]. Available: https://doi.org/10.7910/DVN/3QBYB5

[26] "Log Files - Apache HTTP Server Version 2.4." [Online]. Available: https://httpd.apache.org/docs/2.4/logs.html

[27] A. Agarwal, "Predicting the next search keyword using Deep Learning | by Atul Agar…," May 2023. [Online]. Available: https://archive.ph/vGwEV

An Overview of the 802.11ax Standard

Veronika Bauer, Jonas Andre*, Leander Seidlitz*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: veronika_andrea.bauer@tum.de, andre@net.in.tum.de, seidlitz@net.in.tum.de

***Abstract*—By introducing the WLAN standard 802.11ax, IEEE adjusted Wi-Fi for an era with more users and more connected devices than ever. New developments regarding the Internet of Things, as well as an increasing number of high-density networks, required major improvements compared to the previous standard 802.11ac. Also called High Efficiency Wi-Fi 6, 802.11ax enables more users than ever to be connected to one single access point using new key technologies like OFDMA and MU-MIMO. Wi-Fi 6 also enhances large distance transmissions and improves transmission quality. This paper gives an introduction to the most prominent technologies introduced by Wi-Fi 6 and examines these changes in comparison to its predecessor Wi-Fi 5. Nonetheless, there are several shortcomings of Wi-Fi 6 that will likely be addressed by its successor Wi-Fi 7, which is to be released in 2024.**

***Index Terms*—802.11ax, Wi-Fi 6, High Efficiency Wi-Fi, ODFMA, BSS-Coloring, MU-MIMO**

1. Introduction

Wireless local area networks (WLANs) and cellular networks are becoming increasingly popular. Cisco [1] recently estimated that in 2023, three times more devices were connected to IP networks than there are humans on earth. Compared to 2018, 15 percent more humans will have access to the Internet by the end of 2023, as a study [1] has shown.
With the increasing number of Internet users, the strain on Wi-Fis also amplifies. Especially high-density wireless networks face challenges. Users continue to ask for higher performance and improved user experience. As a response to these challenges, the Institute of Electrical and Electronics Engineers (IEEE) has developed 802.11ax, also known as High Efficiency Wi-Fi or Wi-Fi 6. [2]
Introduced in October 2018, Wi-Fi 6 includes several key technologies crucial to serving a larger number of devices and users in high-density networks. [3] Therefore, two new key technologies are introduced: Orthogonal Frequency Division Multiple Access (OFDMA) and uplink Multi-User Multiple Input Multiple Output (MU-MIMO). Both technologies enable many stations to transmit to the same access point (AP) and vice versa. [4] OFDMA splits a channel into multiple resource units, which can then be employed by numerous stations simultaneously. [4] MU-MIMO uses spatial streams to allow simultaneous transmissions. [4]
Furthermore, Wi-Fi 6 aims to enhance transmissions over larger distances and improve transmission quality by introducing new technologies like midambles, which are symbols added to a transmission for noise reduction. [5]
Compared to its predecessor 802.11ac, also known as Wi-Fi 5, Wi-Fi 6 comes with many new features regarding Physical and MAC[1] Layers. [2]
Therefore, we will present the changes regarding the Physical Layer in Section 2 and MAC Layer changes in Section 3. The new features will be explained one by one. Section 4 summarizes these features and Section 5 briefly introduces Wi-Fi 6's successor Wi-Fi 7, which is supposed to be released in 2024. [7] While this paper provides only a short explanation of the new changes, a more in-depth description can be found in the work by Sankaran et al. [5]. A very recent overview is presented in the work by Natkaniec et al. [8]. For more in-depth information on the two most prominent features, MIMO and OFDMA, readers are referred to the publications by Yang et al. [9] and Lanante et al. [10].

2. Physical Layer

This section presents the changes introduced by 802.11ax regarding the Physical Layer. First, changes made to enable the implementation of OFDMA and MU-MIMO will be explained, namely Resource Units, spacing and symbol duration, and newly introduced frame formats. Following, the two modulation schemes, Quadratic Amplitude and Dual Carrier modulation, will be introduced. Then, two new features targeting Internet of Things use cases, namely Target Wake Time and Midambles, are presented. In the last subsection, we elucidate changes regarding the frequency range, data rate, and guard band.

2.1. Resource Unit Allocation

As one major focus of 802.11ax lies on enabling high-density networks, resource unit allocation helps deal with large numbers of devices connected to the same access point: In Wi-Fi 6, the channel bandwidth is divided by frequency and time. This results in time-frequency blocks called Resource Units (RU). An access point can assign RUs to a particular user. Thus, several stations can transmit simultaneously without collisions. This RU allocation provides the basis for OFDMA. [5]

1. Please note that the extended ISO/OSI model separates the Data Link Layer into two sublayers, the Media Access Control (MAC) Layer and the Logical Link Control (LLC) Layer. [6]

 doi: 10.2313/NET-2024-04-1_08

2.2. Spacing and Symbol Duration

In order to support the use of these Resource Units, Wi-Fi 6 has a four-times decrease in the interval length between subcarriers combined with a four-fold increase in the duration of transmitted symbols. [11] The shortened subcarrier interval increases noise sensitivity. To counter this, symbol duration and guard intervals were increased. [12], [13]
Guard intervals are defined intervals that are used to preserve orthogonality between subcarriers. If chosen too short, intersymbol and intercarrier interference is possible. [14] To avoid these negative consequences, guard intervals and symbol duration have been adjusted in Wi-Fi 6, making the channel more tolerant to jitter, increasing its robustness and transmission efficiency as well as enhancing throughput. Additionally, less bandwidth is wasted. [12], [13]

2.3. Frame Formats

To enable OFDMA and MIMO, the Wi-Fi 6 standard also includes four different physical layer conformance procedure protocol data unit frame formats (PPDU frame formats) [5], [8], [13]:

- Single User PPDU: used for transmissions between only two stations. Wi-Fi 5 already included this PPDU.
- Extended Range Single User PPDU: for transmissions over large distances between two stations. This PPDU can only be used without MIMO, as its main focus is to ensure reliability over a long distance.
- Multi User PPDU: for downstream transmissions with many users involved. This PPDU is needed when using downlink MU-MIMO and MU-OFDMA.
- Trigger based PPDU: used for upstream transmissions involving many users, particularly for uplink MU-MIMO and MU-OFDMA.

2.4. QAM Modulation

To increase the maximum data rate, 1024 Quadratic Amplitude Modulation (QAM) has been introduced. It focuses on use cases with very high channel qualities where data is transmitted over short distances. Using this modulation, ten bits can be transmitted simultaneously. This leads to a potential increase of the available data rate by 25 percent compared to the maximum data rate offered by Wi-Fi 5. However, such a high modulation is susceptible to noise and cannot be applied in low-quality channels or over larger distances. [5], [12], [13]

2.5. Dual Carrier Modulation

1024 QAM cannot be used across large distances. Therefore, dual carrier modulation has been included in Wi-Fi 6. Two subcarriers are modulated using the same information to create redundancy. The receiver can use this to deal with decoding errors. This increases the overall connection robustness and performance across larger distances. Using dual carrier modulation comes at a cost: Only half of the data rate is available, and the modulation can only be utilized for the transmission of Single User PPDUs. [5], [12]

2.6. Target Wake Time

As stated by Sankaran et al. [5], one of the main goals of Wi-Fi 6 was not only to increase throughput but also have a steady power demand compared to its predecessor Wi-Fi 5. The target wake time (TWT) can be used for power saving. Stations can conduct a wake schedule agreement with their access point. They are only awake when necessary and otherwise in power safe mode. Using this feature, a station and an access point agree on a TWT session period, defining the time period when the station has to be awake. Within this time frame, the station can send and receive data. If a station is not needed, it can go to sleep. This reduces congestion of the used medium and saves the station's battery power. [5]
Therefore, this feature is handy when paired with Internet of Things devices. Multiple devices can be connected to a network without increasing congestion and decreasing their battery lifetime. [15]

2.7. Midamble

This feature has been specifically designed for use cases where a connected object, e.g., a drone, moves with up to 60 km/h. A midamble is a high-efficiency long training field (HE-LTF) symbol that can be added to the PPDU payload. Using this information, a receiver can improve its channel estimate. A channel estimate describes the properties of the used channel, like the produced noise or distortion. These properties can then be used to remove unwanted noise and make the connection more reliable. [5], [16]

2.8. Other Changes

Wi-Fi 6 also includes several other noticeable changes on the Physical Layer [13], [17]:

- Frequency range: While Wi-Fi 5 only offers 5 GHz, Wi-Fi 6 includes 2.4 and 5 GHz. In combination with 1024 QAM, this increases the throughput.
- Maximum data rate: 802.11ax features a maximum data rate of 9.6 Gbps when used with the maximum number of possible MIMO streams. Compared to the 6.9 Gbps offered by 802.11ac, this leads to an approximately 40 percent higher maximum data rate compared to Wi-Fi 5.
- Guard band: Wi-Fi 6 supports guard bands of 0.8, 1.6, and 3.2 µs, whereas its predecessor only supports 0.4 and 0.8 µs.

3. MAC Layer

This chapter presents the technologies introduced by 802.11ax that affect the MAC Layer. First, we explain

three crucial technologies that are implemented in Wi-Fi 6 to allow spatial reuse. Then, the concept of Multi-BSSIDs is presented. Last, the two key features of Wi-Fi 6, Uplink/Downlink MU-MIMO and OFDMA, are elaborated.

3.1. Spatial Reuse

Every station belongs to a basic service set (BSS). If two BSSs are very close, stations might be able to detect signals belonging to the respective other BSS as well. Both access points in both BSS might use the same channel. Especially in high-density networks, this can lead to collisions, reduced data rates, and high congestion. To counter this effect, Wi-Fi 6 implements three distinct changes working together to enable a distinction between transmissions belonging to a station's own BSS (intra-BSS) and a foreign BSS (inter-BSS). This can be used to allow parallel transmissions originating from different BSS. [5], [12]

3.1.1. BSS Network Coloring. According to Sankaran and Gulasekaran [5], especially high-density networks can benefit from this new feature. With network coloring, a PPDU can be classified as either intra-BSS or inter-BSS. If a PPDU is intra-BSS, the PPDU has been sent by a station within the same BSS. Inter-BSS PPDUs originate from stations belonging to a different BSS. By being able to distinguish between intra- and inter-BSS PPDUs, stations and APs can recognize PPDUs they do not have to decode and process. To enable BSS coloring, every BSS gets assigned a so-called BSS color. A BSS color is a number between 1 and 63. This number can either be assigned distributed or centralized. With the centralized distribution, a third party assigns BSS colors to the access points. When distributed assignment is chosen, an AP will select its own color different to those of its neighbors. APs advertise their BSS color in the designated field that is part of a beacon frame. Beacon frames are sent periodically to advertise the presence of a network. If a station notices a collision of BSS colors, it sends its own AP a BSS collision event report frame to inform it about the collision. If the AP then decides to change its BSS color, it can include a color change announcement in its next beacon frame. This ensures that all associated stations are always up to date regarding their BSS color. [5]

3.1.2. NAV Mechanism. As spatial reuse allows to treat inter- and intra-BSS PPDUs differently, the network allocation vector (NAV) also has to be split into an intra-BSS NAV and an inter-BSS NAV. The two NAVs act as timers that prohibit stations from sending while another transmission is still ongoing. [5], [18]
With Wi-Fi 5, only one NAV timer was present. If only one NAV timer would be used in Wi-Fi 6, Contention Free End Control (CF-End) frames could cancel a NAV, leading to a malfunction, as this would affect both inter- and intra-BSS frames. [18] With two NAV timers, stations can distinguish between inter-BSS and intra-BSS transmissions and prevent inter-BSS interference. If the two timers are both zero, a station can lower its back-off counter. [5] CF-End frames only cancel either the inter-BSS NAV or intra-BSS NAV. [18] If at least one timer is unequal to zero, the station has to stay in the idle state to avoid collisions. [5], [18]

3.1.3. Parameterized Spatial Reuse. As explained by Mozaffariahrar et al. [12], parameterized spatial reuse helps to enable parallel transmissions. With Wi-Fi 5, stations were not able to send during inter-BSS transmissions even if its sender was so far away that it would not cause interference. BSS network coloring enables stations to distinguish between inter-BSS and intra-BSS transmissions. Parameterized spatial reuse allows stations to cancel the reception of inter-BSS PPDUs or transmit simultaneously. If an inter-BSS PPDU has a power level below a previously determined threshold, a station can treat the medium as if idle and transmit nonetheless. Furthermore, an AP can include its acceptable interference level in the trigger frame. Neighboring APs from different BSSs can then adopt their BSS such that the valid signals they want to receive do not interfere. [5], [12]

3.2. Multi-BSSID

According to Sankaran et al. [5], enhanced multi-BSSID advertisement (EMA) has been developed to reduce the overhead created by management frames. Before Wi-Fi 6, every AP was required to answer probe request frames with beacon frames and also send such a frame on a regular basis. This led to problems when multiple APs were within a very close range which is often the case in high-density environments like stadiums. Every time a new device sends a probe request frame, every AP in range has to answer, leading to a large overhead. With the newly introduced EMA mode, an AP can distinguish between transmitted and non-transmitted basic service set identifiers (BSSIDs) and only answer probe responses for transmitted BSSIDs. Information about non-transmitted BSSIDs is attached to the transmitted BSSID probe response and beacon frames. This aggregation reduces management frame overhead in Wi-Fi 6.

3.3. UL/DL MU-MIMO

Uplink/Downlink multi-user multiple-input multiple-output (UL/DL MU-MIMO) is one of the new key technologies of Wi-Fi 6. [5], [12]
Using downlink MIMO, APs can transmit to numerous stations simultaneously. This works by transmitting a PPDU that combines physical layer conformance procedure service data units (PSDUs) for multiple stations. Each PSDU is transmitted on a different spatial stream. A field in the header of the PPDU indicates the association identifier of the respective station and its dedicated spatial stream. APs can choose new recipients and their streams for every PPDU, fostering flexibility. This feature was already included in Wi-Fi 5, although Wi-Fi 6 now supports up to eight spatial streams. [5], [12]
The newly introduced feature uplink MU-MIMO follows a similar principle. In high-density networks, collisions occur frequently. Although there are mechanisms like request-to-send-clear-to-send (RTS-CTS) protection that aim to prevent collisions, they still occur regularly. Uplink MU-MIMO aims to counter this by ensuring that numerous stations can transmit at the same time. To arrange

a suitable point in time where all stations can transmit simultaneously, an AP sends a trigger frame. Without this synchronization, an AP could not decode the transmitted data easily. Once received, the AP can collectively decode all spatial streams. [5], [12]

3.4. OFDMA

The second key feature, in addition to UL/DL MU-MIMO, is orthogonal frequency division multiple access, short OFDMA. Wi-Fi 5 only offered the usage of orthogonal frequency division multiplexing (OFDM). It is a modulation scheme where a channel is divided into various subcarriers. Transmissions can be done using the subcarriers. This spread takes the whole spectrum and only single users can use OFDM. This increases the collision likelihood in situations with many connected stations, like conference halls or universities. [4], [12]
Orthogonal frequency division multiple access (OFDMA), as presented by Wi-Fi 6, provides a new approach to enable simultaneous transmissions. It also enables higher throughput in high-density networks. OFDMA depends on the allocation of RUs to different stations as described in section 2.1. By using RUs, OFDMA only affords part of the spectrum and multiple simultaneous transmissions are possible, making OFDMA one of the major improvements for high-density networks. Furthermore, RUs within the same transmission can have different sizes depending on the transmitted data. [12], [5]
As was the case for MU-MIMO, OFDMA can be distinguished between uplink and downlink multi-user OFDMA. Uplink OFDMA also requires trigger frames. An AP can send this frame to align the transmission of multiple stations. The trigger frame also contains information about which RUs are assigned to which station. Differing from uplink MU-MIMO, the different data do not thwart each other as they are part of different RUs. Once the AP has received the data, it can send a Multi-STA BACK frame to acknowledge all RUs of all stations. [4]
Using downlink OFDMA, an AP can send to multiple stations at the same time. Information about these transmissions does not have to be transmitted via control frames but is included in the PPDU preamble. Again, RUs are assigned to different stations. The reception of the data is then simultaneously acknowledged by the participating stations using uplink OFDMA. The necessary trigger frame has been delivered to the stations as part of the MAC protocol data unit (MPDU) included in the original PPDU. [4]

4. Conclusion

High Efficiency Wi-Fi 6 includes many new features that mostly focus on enhancing user experience in high-density networks. The most prominent features include OFDMA, MU-MIMO, and Spatial Reuse, including BSS Network Coloring.
To enable these key features, many other changes were necessary. For the introduction of OFDMA and MU-MIMO, RUs had to be included, trigger frames and multi-user PPDUs to be developed. [5], [8], [13] To allow a four-times shorter spacing, the symbol duration was decreased by a factor of four. [12], [13] Spatial reuse relies on the introduction of two NAV timers, parameterized spatial reuse and BSS network coloring. With these mechanisms, stations and APs can easily distinguish between inter- and intra-BSS transmissions. [12], [5], [18]
Apart from these changes, 802.11ax focuses on providing higher throughput: 1024 QAM modulation can theoretically increase the data rate offered by Wi-Fi 5 by 25 percent. In case 1024 QAM is not possible, dual carrier modulation can be used to foster connection robustness and performance. [12], [5], [13] Not only with regard to dual carrier modulation, Wi-Fi 6 aims to enhance user experience over larger distances. The Extended Range single-user PPDU is provided to transmit between two stations across larger distances where MIMO cannot be applied. [8], [5], [13] Furthermore, Wi-Fi 6 also introduces Multi-BSSID to reduce management frame overhead in high-density networks where APs can distinguish between transmitted and non-transmitted BSSIDs. [5]
Another focus of Wi-Fi 6 is the Internet of Things: The introduction of a target wake time is crucial, as many smart devices require batteries. [15]
Midamble symbols are added to the PPDU payload to reduce noise and distortion that occur in channels where objects move. [5]
Overall, Wi-Fi 6 has shown many popular improvements compared to its predecessor Wi-Fi 5 and upgraded user experience, especially in high-density networks, while setting the basis for many IoT applications.

5. Outlook

While Wi-Fi 6 has been highly anticipated, real-world deployment has shown some shortcomings. It has been shown, for example, that for single users, Wi-Fi 5 enables a higher throughput over some distances [11] and it has been noted that Wi-Fi 6 only allows one RU to be allocated to one station [19]. To address these issues, IEEE already launched a working group to design its successor shortly after publishing Wi-Fi 6. 802.11be, also known to become Wi-Fi 7, is said to be released in 2024. [7], [19], [20] Several innovations that overcome shortcomings of Wi-Fi 6 have been identified for Wi-Fi 7, with the most important being:

- OFDMA: Access points should act as schedulers that allocate spectral resources. This is supposed to reduce delay. [7], [19]
- QAM: 4096-QAM modulation would allow for 12 bit symbols to be sent at the same time. Compared to the current 1024 QAM, this increases the data rate by 20 percent. [7]
- Resource Units: RUs are supposed to be enhanced by allocation schemes to be able to assign multiple RUs to one station. This is said to enhance network throughput when combined with OFDMA. [7], [19]
- Extended MU-MIMO: Wi-Fi 7 aims to increase the number of possible spatial streams to 16, doubling the currently available maximum number of streams. As this might influence the accuracy of the channel state information negatively, Wi-Fi 7 will likely also include a channel-sounding method. [19]

Up until now, the final draft of Wi-Fi 7 has not been released and the improvements contained are therefore still subject to change. If Wi-Fi 7 manages to live up to its name, Extremely High Throughput, and further facilitates the use of Internet of Things devices [7], [20], 802.11be will lead the way forward to a more digitalized world.

References

[1] Cisco, "Cisco Annual Internet Report (2018–2023) White Paper," 03 2020, [last accessed 03-March-2024]. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Q. Qu, B. Li, M. Yang, Z. Yan, A. Yang, D.-J. Deng, and K.-C. Chen, "Survey and Performance Evaluation of the Upcoming Next Generation WLANs Standard-IEEE 802.11 ax," *Mobile Networks and Applications*, vol. 24, pp. 1461–1474, 2019.

[3] W.-F. Alliance, "Wi-Fi Alliance® introduces Wi-Fi 6," 10 2018, [last accessed 03-March-2024]. [Online]. Available: https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-wi-fi-6

[4] Y. Daldoul, D.-E. Meddour, and A. Ksentini, "Performance evaluation of OFDMA and MU-MIMO in 802.11 ax networks," *Computer Networks*, vol. 182, p. 107477, 2020.

[5] S. G. Sankaran and S. R. Gulasekaran, *Wi-Fi 6: Protocol and Network*. Artech House, 2021.

[6] IEEE, "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture," *IEEE Std 802-2014 (Revision to IEEE Std 802-2001)*, pp. 1–74, 2014.

[7] Y. A. Qadri, Zulqarnain, A. Nauman, A. Musaddiq, E. Garcia-Villegas, and S. W. Kim, "Preparing Wi-Fi 7 for Healthcare Internet-of-Things," *Sensors*, vol. 22, no. 16, p. 6209, 2022.

[8] M. Natkaniec and M. Kras, "An Optimization of Network Performance in IEEE 802.11 ax Dense Networks," *International Journal of Electronics and Telecommunications*, pp. 169–176, 2023.

[9] H. Yang, D.-J. Deng, and K.-C. Chen, "On energy saving in IEEE 802.11 ax," *IEEE Access*, vol. 6, pp. 47 546–47 556, 2018.

[10] L. Lanante and S. Roy, "Performance analysis of the IEEE 802.11 ax OBSS_PD-based spatial reuse," *IEEE/ACM Transactions on Networking*, vol. 30, no. 2, pp. 616–628, 2021.

[11] A. Masiukiewicz, "Throughput comparison between the new HEW 802.11 ax standard and 802.11 n/ac standards in selected distance windows," *International Journal of Electronics and Telecommunications*, vol. 65, no. 1, pp. 79–84, 2019.

[12] E. Mozaffariahrar, F. Theoleyre, and M. Menth, "A survey of Wi-Fi 6: Technologies, advances, and challenges," *Future Internet*, vol. 14, no. 10, p. 293, 2022.

[13] M. Natkaniec, Łukasz Prasnal, and M. Szymakowski, "A Performance Analysis of IEEE 802.11ax Networks," *International Journal of Electronics and Telecommunications*, vol. 66, no. 1, pp. 225–230, Jul. 2023.

[14] P. Patil, M. Patil, S. Itraj, and U. Bombale, "IEEE 802.11n: Joint modulation-coding and guard interval adaptation scheme for throughput enhancement," *International Journal of Communication Systems*, vol. 33, no. 8, Feb. 2020.

[15] E. Tokhirov and R. Aliev, "Analysis of the differences between Wi-Fi 6 and Wi-Fi 5," in *E3S Web of Conferences*, vol. 402, 2023, p. 03020.

[16] R. Govil, "Different types of channel estimation techniques used in MIMO-OFDM for effective communication systems," *Int J Eng Res Technol (IJERT)*, vol. 7, no. 07, pp. 271–275, 2018.

[17] M. Chotalia and S. Gajjar, "Performance Comparison of IEEE 802.11 ax, 802.11 ac and 802.11 n Using Network Simulator NS3," in *International Conference on Computing Science, Communication and Security*. Springer, 2023, pp. 191–203.

[18] F. Wilhelmi, S. Barrachina-Muñoz, C. Cano, I. Selinis, and B. Bellalta, "Spatial reuse in IEEE 802.11 ax WLANs," *Computer Communications*, vol. 170, pp. 65–83, 2021.

[19] E. Khorov, I. Levitsky, and I. F. Akyildiz, "Current status and directions of IEEE 802.11 be, the future Wi-Fi 7," *IEEE access*, vol. 8, pp. 88 664–88 688, 2020.

[20] Á. López-Raventós and B. Bellalta, "Dynamic traffic allocation in IEEE 802.11 be multi-link WLANs," *IEEE Wireless Communications Letters*, vol. 11, no. 7, pp. 1404–1408, 2022.

Link Failure Detection in Computer Networks

Maximilian Brügge, Manuel Simon*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: maximilian.bruegge@tum.de, simonm@net.in.tum.de

***Abstract*—In computer networks, data must be distributed over long distances, making it susceptible to errors. For this reason, there are specifically designed mechanisms in place to facilitate error recovery. This paper presents different mechanisms of failure detection in networks and classifies the approaches with respect to different failures types. Additionally, it discusses recovery mechanisms and how they correlate with these failure types, providing a holistic view of network resilience. This work underscores the importance of both detecting and effectively recovering from network failures to ensure uninterrupted operation and reliability.**

***Index Terms*—failure detection, fault detection, computer networks, mobile networks, wireless networks**

1. Introduction

As the digital landscape continues to evolve, networks have become the backbone of modern communication. These networks are not only made for data exchange but are also important to the functioning of applications across various domains. With their expansion and increased complexity, networks are now more susceptible to various failure types, ranging from minor transport errors, e.g. packet loss, to major disruptions [1]. This vulnerability highlights the importance of robust failure detection and recovery mechanisms. The ability to immediately and accurately identify failures is crucial in maintaining the reliability and efficiency of network systems. The types of failures encountered in these networks are varied, including hardware malfunctions, software bugs, and issues arising from network topology or design.

The focus of this paper is on selected failure detection in computer networks which are also considered reactive approaches. As outlined by Musumeci et al. [2] and Wang et al. [3], *reactive* strategies differ from *proactive* methods that are aimed at prevention. While *proactive* measures aim to prevent service disruptions, *reactive* approaches such as failure detection are imperative for initiating immediate recovery procedures. These procedures, crucial for swiftly repairing or replacing failed components and thus minimizing downtime, play a vital role in ensuring network resilience. This paper explores the methods and strategies employed for detecting failures in networks, categorizing these based on the types of network failures and presents appropriate recovery solutions. We investigate state-of-the-art and evolution of network reliability, underlining the challenges and advancements in this essential field of network management. This exploration contributes to enhancing the understanding and capabilities of network management, establishing the foundation for the successful deployment and operation of future-oriented networks. Specifically, it addresses the critical need for a very high probability of successful transmission, a key objective in 5G and 6G networks [4], ensuring their advanced performance and service continuity.

2. Technical Background

This section primarily focuses on the fundamental hardware components and essential software elements required for constructing a computer network. Additionally, we present various types of failures that can occur within a computer network.

2.1. Nodes

Peterson et al. [5] define a node as any device that connects to the network, like a computer or a router. These devices serve various roles, from running applications to directing data traffic. Any device capable of transmitting or receiving data to or from a network is classified as a node. In the Internet of Things (IoT), a dishwasher, for instance, could also represent a node within a network. Each node in a network has a limited amount of memory and processing capability, which is essential for processing data and throughput. These nodes are connected to the network via an adapter, which is operated through specialized software. The interplay between a node's memory capacity and its processing speed is a key factor in determining the overall efficiency of the network.

2.2. Links

Links are the channels that connect nodes, using mediums like cables or wireless space to transmit data. These media channels carry data in the form of electromagnetic signals across different frequencies. The physical nature of these links, whether they are made of fiber optics or copper wires, plays a significant role in how data is transmitted and at what speed. The design of these links, including their capacity to handle data and the method of encoding information, is fundamental to network functionality [5].

2.3. Cable-Related Failures

Cable-related failures are a special case of link failures which typically involve optical and copper fiber (i.e. Ethernet), respectively. Optical fiber breaks frequently occur

 doi: 10.2313/NET-2024-04-1_09

unintentionally at construction sites [1], [6]. Although these incidents may occasionally be observed by on-site workers, there are instances where such events go unnoticed by anyone [6]. For these cases, a sophisticated and reliable solution is needed. Another failure type related to optical fiber is high loss, which can occur during installation when workers make errors. These may involve improper fiber bending, contamination of connectors, mishandling of tools, or incorrect connections as described by Fernández et al. [7].

2.4. Wireless Network Failures

Wireless networks present a unique set of failure scenarios, distinct from cable-based networks. These failures are primarily influenced by environmental and physical aspects of wireless signal transmission and reception. Interference is a major cause of wireless network failure. External sources like other electronic devices or physical obstructions can disrupt signal transmission, leading to signal degradation or loss as discussed by M. Gast [8]. Additionally, atmospheric conditions such as heavy rain or fog can also impact signal strength as shown by Osahenvemwen and Omatahunde [9]. Moreover, hardware failures in wireless network components such as routers, access points, and base stations can lead to network outages. Unlike wired networks, where issues might be localized to specific cable faults, failures in key wireless components can have a widespread impact, affecting a large number of users.

Network overload is another critical challenge across all network types, impacting both wireless and wired systems. High device density or excessive data traffic can lead to congestion in any network, potentially causing service outages or significant performance reduction. This phenomenon is especially relevant in wireless networks due to inherent constraints such as limited spectrum and susceptibility to interference as described by C. Casetti et al. [10]. They also describe how the network can become congested, leading to service outages or severe performance degradation. This is particularly evident in both mobile and residential wireless networking environments, where the increasing demand for high-bandwidth applications strains the network's capacity.

3. Failure Detection Mechanisms

This section explores mechanisms employed for detecting failures in computer networks. The complexity and critical nature of networks necessitate robust and precise methods to swiftly identify and address issues, ensuring network reliability and efficiency. We delve into several key technologies and methodologies that have been developed to detect, analyze, and localize failures.

3.1. Optical Time-Domain Reflectometry

When a fiber break occurs, it can be noticed that no packets are received by the recipient anymore, while at the same time the node is also not reachable. This raises the question of whether the issue is link or node-related. For monitoring optical fiber, we can use an Optical Time-Domain Reflectometer (OTDR), which is a leading technique for monitoring optical networks and was developed in 1976 by Barnoski and Jensens [11]. They describe how OTDR operates using Rayleigh scattering, where light reflects off particles smaller than its wavelength. Using this method, a pulsed laser sends light into an optical fiber, creating backscattered light that is detected by a photodiode. This involves sending multiple light pulses during a set measurement time and averaging the traces to enhance accuracy and resolution. The choice of pulse width and measurement times involves trade-offs: shorter pulse widths provide higher resolution but may increase noise and reduce signal visibility due to lower emitted power [11]. In telecom networks, technicians often balance these factors, selecting pulse widths that optimize both resolution and signal clarity. After running a trace generation, OTDR devices yield time-series data which can be automatically analyzed or plotted. High peaks or huge losses in the data usually indicate an error at that position in the fiber. Figure 1 showcases an exemplary OTDR trace, highlighting the use of PC and APC connectors, a power splitter, and Optical Network Units (ONU). This trace vividly illustrates the backscattered light patterns generated by the pulsed laser, marked by significant peaks and troughs. These variations are critical for identifying faults along the fiber, demonstrating the OTDR's precision in fault localization.

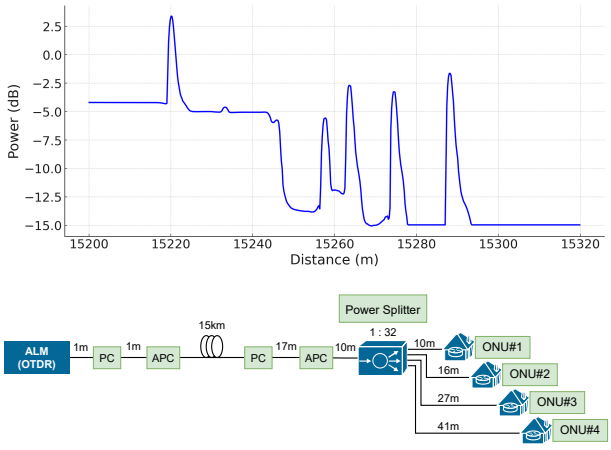


Figure 1: Exemplary OTDR trace using PC and APC Connectors, Power Splitter, and Optical Network Units (ONU)

Advantages of OTDR include its high accuracy and resolution, which allow for precise fault localization. It also enables non-invasive testing, as faults can be detected without physically accessing the entire length of the cable. Additionally, OTDR is versatile, suitable for various fiber types and lengths, and aids in predictive maintenance by detecting potential issues early on. However, there are some disadvantages to using OTDR. As can be seen in Figure 1, the complexity of OTDR traces requires skilled interpretation, especially in networks with intricate architectures. To counteract this, recent advancements have been made in interpreting this data. It has been shown that the characteristics of the different peaks in OTDR traces can be effectively classified using deep learning [12]. This approach significantly enhances the accuracy and

efficiency of fault detection and localization in optical networks.

3.2. Quality of Transmission

Quality of Transmission (QoT) is a metric in evaluating the performance and reliability of optical devices in computer networks. It encompasses the effects of factors such as device ageing and external environmental changes, such as temperature variations, which can progressively degrade transmission quality. Understanding and analyzing QoT is essential for detecting gradual impairments and implementing timely solutions to ensure sustained network efficiency and effectiveness. S. Barzegar et al. [13] discuss two of the most used QoT mechanisms which are discussed in this section.

Bit Error Rate (BER): BER is a measure of the number of bit errors in a transmitted signal. Vela et al. [14] classify BER as a critical metric for assessing the signal quality of optical connections. It becomes particularly relevant in scenarios where signal integrity is compromised due to various types of failures. E.g., signal overlap occurs when an optical connection's spectrum allocation interferes with a neighboring one, often due to inaccuracies in the central frequency of lasers or filters. Tight filtering is another issue, arising from misalignments or inaccuracies in the filter's central frequency or width. Gradual drift happens when the optical signal or filter slowly deviates from its initial central frequency, while cyclic drift is characterized by periodic deviations over time. These failures can lead to sudden or gradual increases in BER. Detecting and analyzing these variations in BER is crucial for identifying and addressing the underlying causes of signal degradation, ensuring the reliability and efficiency of optical network communications. In this context, advanced algorithms, i.e., BANDO and LUCIDA [14], are proposed to detect BER degradation and identify failure patterns, respectively, enhancing the monitoring and management of optical networks.

Signal-to-Noise Ratio (SNR): The level of a desired signal in relation to the background noise level is measured using the SNR. In the work of C. Alkemade et al. [15] the authors delve into the concept of SNR, which they define as the ratio of the power of a signal to the power of the noise and is usually expressed in decibels. A higher SNR indicates a clearer and less noisy signal, making it a crucial parameter in optimizing the performance of communication and detection systems. Figure 2 illustrates the concept of SNR, depicting how signal power compares to noise power in a visual format.

3.3. Received Signal Strength Indicator

Received Signal Strength Indicator (RSSI) is a critical tool used in the realm of wireless networking, particularly in identifying and managing the stability and performance of connections like Wi-Fi access points or mobile networks. According to Y. Chapre et al. [17], it serves as a valuable mechanism for failure detection. RSSI measures the power of signals received by a wireless device. In essence, RSSI measures the signal's strength at a specific location and time. The strength of the received signal is primarily determined by the distance between

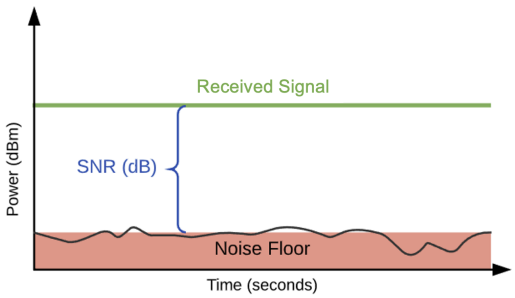


Figure 2: Signal-to-Noise Ratio [16]

the transmitter and the receiver. However, it is not just about distance; various factors can influence RSSI values, making it a dynamic and sensitive measure. For instance, Wi-Fi networks operating in densely populated areas often experience decreased RSSI values due to interference from multiple nearby routers transmitting on the same or overlapping channels. In a typical IEEE 802.11b/g/n network [18], extensive measurements are taken to analyze these factors affecting RSSI. This analysis helps in understanding the reliability and performance of the network under various conditions. The effectiveness of RSSI as a failure detection mechanism lies in its ability to provide a real-time snapshot of the network's signal strength. By continuously monitoring RSSI values, network administrators can quickly identify and address areas with weak signals, ensuring optimal network performance and reliability.

4. Recovery Mechanisms

It is not only crucial to identify network failures but also to address them, ensuring continued network operation. Following the presentation of link failure mechanisms in computer networks, this section shifts focus to recovery techniques and strategies. Rapid recovery is essential to maintain the network's reliability and efficiency. We will discuss various technologies and methods that support seamless network operations even after disruptions.

4.1. Lightpath Re-Routing

Lightpath re-routing in optical networks, particularly those utilizing dense wavelength division multiplexing (DWDM) and optical switches, is a process for ensuring efficient and reliable network operations, as explored by Bouillet et al. [19] in their research. These networks manage service requests through an online routing algorithm that dynamically decides the best routing paths based on current network information. The challenge lies in maintaining service continuity, especially given the high connection rates of these networks, which can reach tens of Terabits per second. Two restoration strategies are employed in these networks: end-to-end dedicated mesh protection and shared mesh restoration. The dedicated mesh approach uses predefined backup paths that are diverse from the primary paths, ensuring that both do not fail simultaneously. This method requires significant capacity, as backup paths are often longer than primary

TABLE 1: Overview of all discussed detection mechanisms

Detection Mechanism	Hardware Level	Failure Types	Recovery Mechanisms	Literature
Optical Time Domain Reflectometry (3.1)	Optical Fiber/Devices	Optical fiber bend, break, loss	Lightpath Re-Routing (4.1)	[11], [12], [19]
Quality of Transmission (3.2)	Link	Signal degradation, noise interference	Lightpath Re-Routing (4.1)	[13]–[15], [19]
Received Signal Strength Indicator (3.3)	Link	Interference, Distance, Obstructions, Malfunction	Dynamic Channel Assignment (4.2)	[17], [18], [20]

paths. However, it offers the advantage of immediate restoration due to the permanence of the backup paths. In contrast, shared mesh restoration allows backup paths to share capacity, provided the primary paths are mutually diverse. This approach saves more space in the network but is slower to recover, as it requires additional steps to establish a backup route. The length and number of hops of the backup path can influence the restoration time, posing a trade-off between cost and recovery latency.

4.2. Dynamic Channel Assignment

In Wireless Local Area Networks (WLANs), a channel refers to a specific frequency range in the radio spectrum for wireless communication. Figure 3 illustrates the 2.4GHz Wi-Fi channel band, highlighting the available channels within this spectrum. Strategically selecting Access Point (AP) channels is crucial for enhancing network performance, especially in dense environments where overlapping frequencies may cause interference. This overlap leads to interference and network congestion, adversely affecting network functionality. The impact of interference is closely monitored using the RSSI (c.f. Section 3.3). Advanced channel assignment algorithms aim to minimize the impact of interference. By focusing on reducing the interference impact, these algorithms can significantly improve data transmission quality and overall network reliability. The effectiveness of advanced algorithms, such as Wi-5, has been demonstrated through real-world evaluations. [20]

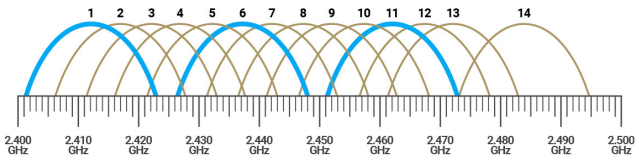


Figure 3: 2.4GHz Wi-Fi Channel Band [21] based on D. Coleman et al. [22]

5. Conclusion and Future Work

This paper has presented an overview of various link failure detection mechanisms in computer networks, emphasizing the significance and applicability of each method under different failure scenarios. The detailed analysis, as summarized in Table 1, provides a clear comparison of these mechanisms, highlighting the involved hardware, failure types and their possible recovery.

The advancements in technologies such as deep learning, as mentioned, offer promising avenues for enhancing the efficiency and accuracy of these detection methods. This paper has underscored the importance of selecting appropriate detection mechanisms based on the specific nature of the potential failures, a decision critical for maintaining the robustness and reliability of optical networks. Future research may focus on the integration of artificial intelligence and machine learning to further refine these detection methods, potentially automating the process and providing more nuanced insights into network health and integrity. The evolving landscape of optical network technology continues to present new challenges and opportunities, making ongoing research and development in this field necessary.

References

[1] Spiegel, "Bauarbeiten offenbar Ursache von IT-Ausfall bei der Lufthansa," Februrary 15, 2023, Accessed: February 29, 2024. [Online]. Available: https://www.spiegel.de/wirtschaft/unternehmen/lufthansa-bagger-kappt-kabel-bauarbeiten-offenbar-ursache-von-it-ausfall-a-bb48b9e6-3c5b-4f22-9726-75de334525bc

[2] F. Musumeci, C. Rottondi, G. Corani, S. Shahkarami, F. Cugini, and M. Tornatore, "A tutorial on machine learning for failure management in optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4125–4139, 2019.

[3] D. Wang, C. Zhang, W. Chen, H. Yang, M. Zhang, and A. P. T. Lau, "A review of machine learning-based failure management in optical networks," *Science China Information Sciences*, vol. 65, no. 11, p. 211302, 2022.

[4] A. Gupta, X. Fernando, and O. Das, "Reliability and availability modeling techniques in 6g iot networks: A taxonomy and survey," in *2021 International Wireless Communications and Mobile Computing (IWCMC)*, 2021, pp. 586–591.

[5] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2007.

[6] M. Illidge, "Construction causes major cable break in Johannesburg — and almost no-one knew," October 11, 2023, Accessed: February 29, 2024. [Online]. Available: https://mybroadband.co.za/news/fibre/510900-construction-causes-major-cable-break-in-johannesburg-and-almost-no-one-knew.html

[7] M. P. Fernández, L. A. B. Rossini, J. P. Pascual, and P. A. C. Caso, "Enhanced fault characterization by using a conventional otdr and dsp techniques," *Opt. Express*, vol. 26, no. 21, pp. 27 127–27 140, Oct 2018. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-26-21-27127

[8] M. Gast, *802.11 wireless networks: the definitive guide*. " O'Reilly Media, Inc.", 2002.

[9] A. O. Osahenvemwen and B. E. Omatahunde, "Impacts of weather and environmental conditions on mobile communication signals," *Journal of Advances in Science and Engineering*, vol. 1, no. 1, pp. 33–38, Apr. 2018. [Online]. Available: http://www.sciengtexopen.org/index.php/jase/article/view/8

[10] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP westwood: end-to-end congestion control for wired/wireless networks," *Wireless Networks*, vol. 8, pp. 467–479, 2002.

[11] M. K. Barnoski and S. M. Jensen, "Fiber waveguides: a novel technique for investigating attenuation characteristics," *Appl. Opt.*, vol. 15, no. 9, pp. 2112–2115, Sep 1976. [Online]. Available: https://opg.optica.org/ao/abstract.cfm?URI=ao-15-9-2112

[12] M. Brügge, J. Müller, S. K. Patri, S. Jansen, J. Zou, S. Althoff, and K.-T. Förster, "Live demonstration of ML-based PON characterization and monitoring," in *2023 Optical Fiber Communications Conference and Exhibition (OFC)*, 2023, pp. 1–3.

[13] S. Barzegar, M. Ruiz, A. Sgambelluri, F. Cugini, A. Napoli, and L. Velasco, "Soft-failure detection, localization, identification, and severity prediction by estimating QoT model input parameters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2627–2640, 2021.

[14] A. P. Vela, M. Ruiz, F. Fresi, N. Sambo, F. Cugini, G. Meloni, L. Potì, L. Velasco, and P. Castoldi, "BER degradation detection and failure identification in elastic optical networks," *Journal of Lightwave Technology*, vol. 35, no. 21, pp. 4595–4604, 2017.

[15] C. Alkemade, W. Snelleman, G. Boutilier, B. Pollard, J. Winefordner, T. Chester, and N. Omenetto, "A review and tutorial discussion of noise and signal-to-noise ratios in analytical spectrometry—i. fundamental principles of signal-to-noise ratios," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 33, no. 8, pp. 383–399, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0584854778800494

[16] Hollyland, "All about signal to noise ratio," https://www.hollyland.com/blog/tips/signal-to-noise-ratio, 10 2023, Accessed: February 29, 2024.

[17] Y. Chapre, P. Mohapatra, S. Jha, and A. Seneviratne, "Received signal strength indicator and its analysis in a typical wlan system (short paper)," in *38th Annual IEEE Conference on Local Computer Networks*, 2013, pp. 304–307.

[18] "IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac)and physical layer (phy) specifications amendment 5: Enhancements for higher throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, 2009.

[19] E. Bouillet, J.-F. Labourdette, R. Ramamurthy, and S. Chaudhuri, "Lightpath re-optimization in mesh optical networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 437–447, 2005.

[20] A. Raschellà, M. Mackay, F. Bouhafs, and B. I. Teigen, "Evaluation of channel assignment algorithms in a dense real world wlan," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–5.

[21] EnGenius Technologies, Inc., "Your Go-To-Guide for Channel & Transmit Power on Wi-Fi Networks (Part 2)," https://www.engeniustech.com/wp-content/uploads/2017/10/blog_nov1.jpg, 10 2017, Accessed: February 29, 2024. [Online]. Available: https://www.engeniustech.com/go-guide-channel-transmit-power-wi-fi-networks-2/

[22] D. D. Coleman and D. A. Westcott, *Cwna: certified wireless network administrator official study guide: exam Pw0-105*. John Wiley & Sons, 2012.

ZDNS vs MassDNS: A Comparison of DNS Measurement Tools

Jeremy Dix, Patrick Sattler*, Johannes Zirngibl*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: jeremy.dix@tum.de, sattler@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—The DNS is an ever-growing essential part of today's Internet. Its ability to function well is crucial for the Internet's stability and maintainability. To ensure that, the DNS needs to incorporate scalable and high-performing technologies. Active DNS measurement on a large-scale basis plays a critical role in developing these technologies. ZDNS and MassDNS are two frameworks that provide toolsets for performing such DNS measurements while taking different approaches to accomplish this task. This paper describes and compares these frameworks and their features in detail and provides an overview of their respective resolution accuracy. Most notably, it shows that both tools incorporate features, e.g., for internal recursion and lookup chain exposure (ZDNS) or subdomain enumeration (MassDNS). It demonstrates that, while both frameworks have a similar accuracy under the same circumstances, MassDNS tends to experience more timeouts in general. The paper aims to help researchers and other interested parties choose the right tool for their use case by better understanding their specific capabilities.**

***Index Terms*—dns, networking, performance, benchmark, measurement, zdns, massdns**

1. Introduction

Nowadays, the World Wide Web is bigger than ever before. According to the Domain Name Industry Brief, it currently comprises about 359.3 million domain names [1] and is exponentially growing each year. The Domain Name System (DNS) is critical for accommodating this vast number of domains and is integral to the Internet's infrastructure. Its primary purpose is to convert human-readable domain names into IP addresses.

Due to its fast growth [1], the DNS requires highly performing, scalable, and secure technologies to keep up with the increasing demand. In particular, the field of active DNS measurement is crucial for conducting the necessary research to create these technologies. For example, a research team at the University of Twente published a paper that describes the importance of extensive active DNS measurement and its challenges, e.g., the burdens imposed on the DNS by daily scans [2].

For conducting research involving active DNS measurement, there is a need for high-performance tools that can perform domain name resolution on a large scale. Two such recently developed tools that are actively being used for DNS research are ZDNS and MassDNS.

ZDNS is a command line tool implemented in Go [3], created by Durumeric et al. at Stanford University as part of the ZMap Project [4]. In contrast, MassDNS is a single-threaded DNS stub resolver written in C, created by Birk Blechschmidt, a security engineer at Deutsche Telekom, and Quirin Scheitle, a former researcher at the Technical University of Munich [5].

Comparing these two tools is particularly interesting, as their feature set, purpose, and commitment to delivering high resolution speeds are very alike. MassDNS, for instance, promises a speed of up to 350.000 lookups per second [5]. However, while these toolsets have the same primary purpose, they still differ in functionality, usability, resolution accuracy, and scalability. Depending on the use case, this might lead to one tool performing better. Hence, this paper compares both tools w.r.t. the latter properties to determine their suitability for specific use cases.

2. Related Work

ZDNS and MassDNS mainly contribute to the field of active DNS measurement. Joint research projects like TIDE [6] or OpenINTEL [7] are significant contributors to this research area as well. OpenINTEL measures more than 252 million domains every day, thus precisely logging the state of the DNS over time [7]. The University of Twente, as well as research organizations like SIDN Labs [8], SURFnet [9], and NLnet Labs [10] are all major contributors to OpenINTEL, as well as other active DNS measurement research projects (e.g., [2], [11], [12]). Notably, another DNS measurement platform, Censys Search [13], uses ZDNS for its DNS measurements [3]. MassDNS is also used in various DNS measurement research projects, e.g., [14] or [15].

The ZDNS developers themselves published a paper in 2022 in which ZDNS is compared to other DNS tools [3]. The paper explains ZDNS' properties, highlights its performance in large-scale domain resolution, and indicates possible use cases.

Apart from ZDNS and MassDNS, several other tools exist that were created by research teams for internal measurements in the context of various research projects [2], [16]. Additionally, there are several Linux tools with similar purposes, e.g., domain information groper (`dig`) [17], or `nslookup` [18]. However, in contrast to the frameworks presented in this paper, these solutions/tools are generally not published, preventing other scientists from using them in their research projects, or they are not optimized for large-scale DNS measurements.

Lastly, tools like ZDNS and MassDNS typically rely on third-party recursive resolvers, like Unbound [19] or the Cloudflare public resolver (1.1.1.1) [20] to resolve

 doi: 10.2313/NET-2024-04-1_10

domain names. These resolvers provide the necessary infrastructure for communication with the name servers in the DNS.

3. General Comparison

This chapter gives a comprehensive overview of both tools' features, compares their usability and implementation, and lists some of their approaches to fault mitigation. Table 1 summarizes both tools' most significant properties.

TABLE 1: Feature and property overview.

Feature	ZDNS	MassDNS
Number of fully supported record types	70	11
Supported L4 protocols	UDP, TCP	UDP
Concurrency method	Goroutines	epoll
Built-in resolver	✓	-
Iterative lookups	✓	✓
Lookup chain exposure	✓	-
Additionally queried record types	`A`, `CNAME`	`PTR`
Native subdomain enumeration support	-	✓
JSON output	✓	✓

3.1. Feature Overview

Both tools serve the purpose of resolving domain names by querying various resource record types. However, while ZDNS fully supports a total of 70 DNS record types [21], MassDNS only supports the 11 most commonly used types for its human-readable output formats: `A`, `AAAA`, `CAA`, `CNAME`, `DNAME`, `MX`, `NS`, `SRV`, `PTR`, `SOA`, and `TXT` [5].

Moving on, ZDNS features its own standalone caching recursive resolver library [22], which is built upon the DNS library by Miek Gieben [23]. It consists of all DNS resource record types, including the records from the DNS Security Extensions (DNSSEC) [22].

ZDNS' Built-in Resolver. While MassDNS exclusively relies on third-party recursive resolvers to perform DNS queries, ZDNS also has a built-in iterative resolver. It makes use of the aforementioned library for its operation. According to the developers of ZDNS, an advantage of this resolver is its ability to circumvent potential rate limits by public resolvers when querying with high concurrency. This resolver also performs local recursion, which exposes internal DNS procedures to the user, enabling them to conduct more detailed research. Moreover, the resolver includes a selective response cache. It minimizes the cache entry size by only buffering Name Server (`NS`) and glue records. By leveraging this technique, it prevents excessive disposal of cache entries while reducing the number of needed queries and helping the resolver with subsequent recursion. [3]

Still, when comparing the performance of ZDNS' iterative resolver to public resolvers like Cloudflare, it can be seen that public resolvers are better suited for large-scale lookups of billions of domains. This is due to bigger caches, and thus better resolution accuracy with bigger scale [3]. Therefore, for large-scale lookups, both tools depend on external recursive resolvers to reach their full potential.

ZDNS' Lookup Modules. ZDNS encompasses several *lookup modules*, namely `mxlookup` and `alookup` [21]. These modules can automatically perform additional queries when processing Mail Exchange and Canonical Name records in addition to looking up the records specified in the original query. The `mxlookup` module is able to automatically perform additional `A` record lookups when querying for `MX` records, while the `alookup` module can interpret `CNAME` records and query their contents.

MassDNS' Python Scripts. MassDNS has several distinctive features as well. A couple of Python scripts expand the use cases of MassDNS, particularly with regard to reconnaissance scanning. For instance, a script for automatically resolving previously queried PTR records named `ptr.py` enables the user to efficiently perform Reverse DNS [24].

In addition, the script `subbrute.py` allows for brute-force subdomain enumeration with MassDNS [5]. It works similarly to SubBrute, a high-performance DNS query spider, which can perform DNS record and subdomain enumeration [25]. MassDNS includes several more subdomain enumeration scripts, e.g., the script `ct.py` which uses the crt.sh Identity Search to scrape for Certificate Identity logs and extract subdomains from them [26]. These scripts present an advantage of MassDNS, as their functionality in enumerating subdomains is useful for reconnaissance scans and penetration tests, which are performed for detecting potential security risks or vulnerabilities of domains.

3.2. Usability Comparison

Both DNS frameworks provide the user with various CLI options for setting basic DNS lookup parameters, like the timeout time and query retry count.

ZDNS additionally has the option to use the built-in iterative resolver via setting the `--iterative` flag, but also an option to enable UDP socket recycling (see Section 3.3), as well as options for choosing specific transport layer protocols or *lookup modules*. [21]

In contrast, most notably, MassDNS has an option to unset the DNS Recursion Desired (RD) bit via setting the `--norecurse` flag [5]. This option enables the user to perform non-recursive lookups, and thus makes it easier to carry out DNS cache snooping [27] or subdomain enumeration (see Section 3.1).

Output. Further considering the output, there are several similarities and differences in its format and verbosity. MassDNS incorporates a total of 5 different output formats: domain list output, simple text output, full text output, binary output, and Newline Delimited JSON (NDJSON) output [5], [28]. The simple text output can be controlled by a total of 10 advanced options for high customization [5]. It is important to note that out of these output options, only the binary output conserves the whole DNS response data consisting of all received records [5]. All other formats only preserve the record types mentioned in Section 3.1. Most notable, however, is the JSON output format. It outputs all response packets as JSON objects in the output style of `dig`, which are easily programmatically interpretable, thus enabling researchers to parse their results efficiently.

ZDNS, on the other hand, only features output in JSON, whereby different output verbosities exist: `short`, `normal`, `long`, and `trace`. These verbosities can be further customized through the `--include-fields` option. [21]

The ZDNS' `trace` feature is particularly interesting, as it can display the whole lookup chain/trace of every request when performing internal recursion. It lists the responses of all name servers of the trace, starting from the root name server [3]. The `dig` tool has a similar feature, which can be used via the `+trace` argument [17]. MassDNS, on the other hand, lacks an analogous feature.

In conclusion, although MassDNS supports more output options than ZDNS, the most significant output variant for further processing, namely the JSON output method, is supported by both tools. However, in ZDNS, this output format is customizable, whereas in MassDNS, it is not.

3.3. Implementation Differences

The performance and reliability of each tool highly rely on its implementation. For this reason, key implementation aspects of both tools are compared below.

ZDNS and MassDNS both rely on different forms of concurrency when executing queries. ZDNS uses Goroutines, i.e., lightweight threads, for looking up names in parallel. Each Goroutine is assigned a UDP socket, which sends and receives DNS packets. By default, ZDNS reuses its UDP sockets, meaning they are used repeatedly for different lookups until program termination. This improves ZDNS performance and scalability, as the creation of new sockets for each new query is circumvented. [3]

MassDNS, on the other hand, does not use threads in its C implementation. It instead utilizes the `epoll` I/O event notification facility together with a self-written hashmap implementation to carry out concurrent queries [29], [5]. MassDNS creates one UDP socket per `--bind-to` entry and then assigns all sockets to the `epoll` instance [5]. The hashmap respectively stores the domain name queries that are supposed to be processed next, so that the `epoll` instance can take the next batch of domain names and use an available socket to execute the lookups. Additionally, MassDNS can be run utilizing multiple processes. However, it then operates using a shared-nothing architecture [5], meaning that queries are not synced between processes, and the outputs of each process are stored in different files.

As mentioned above, both tools utilize UDP sockets for their lookups, but ZDNS also allows for the use of TCP sockets. A disadvantage of using the latter is that it negatively affects performance, as TCP sockets do not allow for reusability. [21]

3.4. Fault Mitigation Approaches

ZDNS and MassDNS share one big issue; they both can overwhelm DNS servers and experience rate limiting by public resolvers when run with high concurrency. To resolve this issue, both tools provide an option to use multiple resolvers for querying.

ZDNS offers this feature via its `--name-servers` option [21]. However, as described in Section 3.1, the latter issue can be avoided entirely by using ZNDS' built-in resolver.

MassDNS offers a similar feature to use multiple resolvers via its `--resolvers` option [5]. Besides that, for dodging IPv6 resolver rate-limiting, MassDNS provides the `--rand-src-ipv6 <your_ipv6_prefix>` option [5]. This option allows MassDNS to pick from a specified subnet of source IPv6 addresses for each lookup, thus effectively evading rate limits by some IPv6 resolvers. Other than that, there is no other option to minimize the load on DNS servers when using MassDNS except to manually decrease the hashmap size or lower the retry count from the default 50 [5].

4. Performance Evaluation

The performance of each tool in successfully resolving domain names is crucial for assessing the capabilities of both frameworks. It is evaluated by comparing each tool's resolution accuracy and fallibility. Furthermore, the Top Level Domains (TLDs) of the domain names resolved exclusively by one of the tools, but not the other, are compared to demonstrate either tool's effectiveness when querying for domain names with specific TLDs.

4.1. Resolution Accuracy Comparison

The resolution accuracy of ZDNS and MassDNS was compared by querying for `A` and `AAAA` records of a custom dataset containing all domain names from the Cloudflare (CF) Radar Top 1 Million (1 009 126 domain names on 11-30-2023) [30] and the Chrome User Experience Report (CrUX) (995 207 distinct domain names on 11-30-2023) [31]. These domain lists were chosen because they are both publicly available and contain the most visited, i.e., most resolved domains of the Internet.

Both tools were tested once using the Unbound recursive resolver, while ZDNS was also tested using the CF public resolver. The CF resolver was additionally chosen for ZDNS, as, according to the ZDNS developers' scans, ZDNS overall works best when paired with CF [3]. The built-in ZDNS iterative resolver was not tested, as MassDNS does not feature a native resolver, thus rendering a comparison inequitable.

Inherently, MassDNS uses 10 000 concurrent lookups. However, due to ZDNS only using 1000 Goroutines by default [21], MassDNS was tested with a hashmap size of both 10 000 and 1000 to provide a more balanced comparison. To minimize resolver overload, ZDNS was tested using its default retry count of 1, while MassDNS' retry count was lowered from 50 to 3.

Success Rates. The success rates discussed below were achieved by ZDNS and MassDNS using the Unbound resolver and a concurrency level of 1000 lookups. Figure 1 depicts the encountered error codes while querying for `A` records using the aforenamed parameters.

First, considering the CF Radar query results, it became apparent that, regardless of whether `A` or `AAAA` records were queried, MassDNS had a slightly higher resolution success rate than ZDNS. For instance, MassDNS successfully resolved 96.54% of `A` record queries, whereas ZDNS resolved 96.51% of queries with success. In the CrUX scan results, while ZDNS excelled in resolution accuracy this time, the success rates of both tools were also quite similar, but they both encountered a higher

 doi: 10.2313/NET-2024-04-1_10

number of errors. In the case of A records being queried, MassDNS had a slightly lower success rate of 95.39%, while ZDNS marginally outperformed MassDNS with a success rate of 95.49%.

In conclusion, the results show that, under equivalent circumstances, both tools have a similar accuracy. Which tool gets a slight advantage depends on the queried domain list. However, as seen in Section 4.2, there is a major disparity in accuracy when tweaking the tools' parameters.

Error Comparison. Examining the status/error codes in Figure 1 reveals some more interesting aspects. When scanning the CrUX list, both tools encountered about twice as many SERVFAILS and NXDOMAIN errors as they did while querying the CF Radar list.

Concerning TIMEOUT encounters, each time, MassDNS experienced more timeouts than ZDNS. For instance, while resolving the CF Radar A records, MassDNS experienced 988 more timeouts than ZDNS. The timeout difference is even more significant when looking at the CrUX A record lookup results, where it amounts to 1420. Considering that MassDNS only timed out 4973 times in this scan, this represents a timeout increase of about 28.55% when compared to ZDNS.

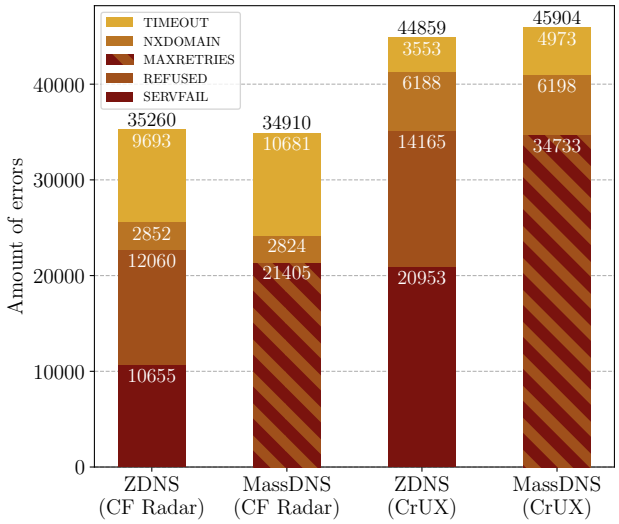


Figure 1: Comparison of A record query errors. The MassDNS specific MAXRETRIES status code encompasses SERVFAIL and REFUSED errors.

Resolved Domain Name Comparison. The domain names that both tools resolved successfully were mostly similar. For both domain lists, the similarity percentage when querying A records amounted to about 99.76%.

However, when analyzing the domain names each tool resolved exclusively, some noteworthy aspects became visible. Table 2 depicts the TLDs where ZDNS and MassDNS exhibited the most notable disparities in successful resolutions. The table shows that ZDNS performed slightly better when resolving .com domain names. The tool resolved 853 domains which MassDNS could not successfully query. In fact, ZDNS excelled when resolving domain names for all TLDs listed in Table 2 except when querying .ru domains. In this case, MassDNS outperformed ZDNS by uniquely resolving an additional 569 domain names.

TABLE 2: Number of TLDs either tool resolved exclusively across all A record queries.

TLD	ZDNS	MassDNS
.com	853	352
.ru	223	792
.jp	589	153
.net	199	61
.info	130	55
.club	128	39
Total	2622	1929

4.2. Additional Findings

ZDNS paired with CF. Scanning the aforementioned domain lists (see Section 4.1) with ZDNS in combination with the CF public resolver revealed that ZDNS performs more effectively with it than with the Unbound resolver. For instance, the success rate rose from 96.54% to a near-perfect 99.26% when querying the CF Radar list for A records. This result underlines that ZDNS is more performant when paired with Cloudflare [3].

Timeouts in MassDNS. Running the previously presented scans with MassDNS using its default hashmap size of 10K revealed an interesting aspect regarding the times MassDNS experienced timeouts. For example, when scanning the CF Radar list for A records, MassDNS encountered 72 148 timeouts when using 10K concurrent lookups, which are 61 467 more timeouts than in the scan presented in Figure 1. In other words, MassDNS encountered around six times as many timeouts when performing 10K concurrent queries compared to only 1000 parallel lookups. A similar outcome was observed when scanning the CrUX list.

This behavior implies that MassDNS overwhelms name servers when querying with its default concurrency, which leads to timeouts that stem from rate limits. This notably adversely affects MassDNS' success rate in every scan. For instance, when scanning the CF Radar for A records, the success rate decreased from 96.51% to 90.66%. According to Durumeric et al., MassDNS' success rate drops even further to about 65% when using 45K concurrent lookups [3].

5. Conclusion and Outlook

ZDNS and MassDNS are both capable tools for pure large-scale domain name resolution.

ZDNS is highly preferable when reliability and extensibility are the priority. Its main advantage is that it offers a well-rounded, feature-rich package. Different properties, such as support for numerous record types, advanced DNS lookup modules, and a built-in resolver for, e.g., analyzing lookup chains, make it a versatile instrument suitable for intricate DNS analysis.

MassDNS is a more fitting choice if runtime takes priority over reliability. Its exceptionally high default concurrency rate renders it highly suitable for fast bulk queries. However, MassDNS' extensive parallelism comes at the expense of reduced reliability as the timeout rate increases. Nevertheless, MassDNS offers a variety of use cases, especially in the fields of reconnaissance scanning,

penetration testing, and Reverse DNS. The Python scripts that MassDNS features are highly beneficial for applications in these areas.

Both tools plan to implement new features and combat their shortcomings in the future. While ZDNS plans to extend its functionality by adding support for DNS over HTTPS and DNS over TLS [3], MassDNS aims to implement more adaptable concurrency mechanisms to prevent overwhelming resolvers, as well as more reconnaissance features, e.g., wildcard record detection [5]. These improvements will make both tools even more suitable for their respective use cases.

References

[1] DNIB, "The domain name industry brief quarterly report," https://dnib.com/articles/the-domain-name-industry-brief-q3-2023, Tech. Rep., 2023, [Online; accessed 1-March-2024].

[2] R. van Rijswijk-Deij, M. Jonker, A. Sperotto, and A. Pras, "A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1877–1888, 2016.

[3] L. Izhikevich, G. Akiwate, B. Berger, S. Drakontaidis, A. Ascheman, P. Pearce, D. Adrian, and Z. Durumeric, "ZDNS: A Fast DNS Toolkit for Internet Measurement," in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 33–43. [Online]. Available: https://doi.org/10.1145/3517745.3561434

[4] The ZMap Team, "The ZMap Project," https://zmap.io/, 2024, [Online; accessed 1-March-2024].

[5] B. Blechschmidt and Q. Scheitle, *MassDNS*, https://github.com/blechschmidt/massdns, [Online; accessed 1-March-2024].

[6] S. Meng, L. Liu, and V. Soundararajan, "Tide: Achieving Self-Scaling in Virtualized Datacenter Management Middleware," in *Proceedings of the 11th International Middleware Conference Industrial Track*, ser. Middleware Industrial Track '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 17–22. [Online]. Available: https://doi.org/10.1145/1891719.1891722

[7] OpenINTEL, "Openintel," https://openintel.nl/, 2024, [Online; accessed 1-March-2024].

[8] SIDN Labs, "About SIDN Labs," https://www.sidnlabs.nl/en/about-sidnlabs, 2024, [Online; accessed 1-March-2024].

[9] SURF, "About SURF," https://www.surf.nl/en/about, 2024, [Online; accessed 1-March-2024].

[10] NLnet Labs, "About," https://nlnetlabs.nl/about/, 2024, [Online; accessed 1-March-2024].

[11] R. Sommese, M. Jonker, J. van der Ham, and G. C. M. Moura, "Assessing e-Government DNS Resilience," in *2022 18th International Conference on Network and Service Management (CNSM)*, 2022, pp. 118–126.

[12] R. Sommese, G. C. M. Moura, M. Jonker, R. van Rijswijk-Deij, A. Dainotti, K. C. Claffy, and A. Sperotto, "When Parents and Children Disagree: Diving into DNS Delegation Inconsistency," in *Passive and Active Measurement*, A. Sperotto, A. Dainotti, and B. Stiller, Eds. Cham: Springer International Publishing, 2020, pp. 175–189.

[13] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 542–553. [Online]. Available: https://doi.org/10.1145/2810103.2813703

[14] J. Zirngibl, P. Sattler, and G. Carle, "A First Look at SVCB and HTTPS DNS Resource Records in the Wild," in *2023 IEEE European Symposium on Security and Privacy Workshops*, Jul. 2023.

[15] L. Degani, F. Bergadano, S. Mirheidari, F. Martinelli, and B. Crispo, "Generative adversarial networks for subdomain enumeration," 04 2022, pp. 1636–1645.

[16] J. Mao, M. Rabinovich, and K. Schomp, "Assessing Support for DNS-over-TCP in the Wild," in *Passive and Active Measurement: 23rd International Conference, PAM 2022, Virtual Event, March 28-30, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 487–517. [Online]. Available: https://doi.org/10.1007/978-3-030-98785-5_22

[17] Internet Systems Consortium, Inc., *dig - Linux man page*, https://linux.die.net/man/1/dig, [Online; accessed 1-March-2024].

[18] A. Cherenson, *nslookup - Linux man page*, https://linux.die.net/man/1/nslookup, [Online; accessed 1-March-2024].

[19] NLnetLabs, *Unbound*, https://github.com/NLnetLabs/unbound, [Online; accessed 1-March-2024].

[20] Cloudflare, "What is 1.1.1.1?" https://www.cloudflare.com/learning/dns/what-is-1.1.1.1/, 2024, [Online; accessed 1-March-2024].

[21] Z. Durumeric, P. Pearce, D. Adrian, S. Kalscheuer, and B. VanderSloot, *ZDNS*, https://github.com/zmap/zdns, [Online; accessed 1-March-2024].

[22] B. Blechschmidt and Q. Scheitle, *Alternative (more granular) approach to a DNS library (fork)*, https://github.com/zmap/dns, [Online; accessed 1-March-2024].

[23] M. Gieben, *Alternative (more granular) approach to a DNS library*, https://github.com/miekg/dns, [Online; accessed 1-March-2024].

[24] Cloudflare, "What is reverse DNS?" https://www.cloudflare.com/learning/dns/glossary/reverse-dns/, 2024, [Online; accessed 1-March-2024].

[25] The Rook, *subdomain-bruteforcer (SubBrute)*, https://github.com/TheRook/subbrute, [Online; accessed 1-March-2024].

[26] R. Stradling, "crt.sh Certificate Search," https://crt.sh/, [Online; accessed 1-March-2024].

[27] O. Farnan, J. Wright, and A. Darer, "Analysing Censorship Circumvention with VPNs Via DNS Cache Snooping," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 205–211.

[28] T. Hoeger, C. Dew, F. Pauls, and J. Wilson, *NDJSON - Newline delimited JSON*, https://github.com/ndjson/ndjson-spec, [Online; accessed 1-March-2024].

[29] *epoll - Linux man page*, https://linux.die.net/man/7/epoll, [Online; accessed 1-March-2024].

[30] Cloudflare Radar, "Top 1000000 Domains," https://radar.cloudflare.com/domains, 2023, [Online; accessed 30-November-2023].

[31] Google, "Overview of CrUX," https://developer.chrome.com/docs/crux/, 2023, [Online; accessed 30-November-2023].

Survey on Recent Applications of Extreme Value Theory in Networking

Jana Nina Friedrich, Max Helm*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: jana.friedrich@tum.de, helm@net.in.tum.de

***Abstract*—The statistical model Extreme Value Theory (EVT) predicts extreme events, e.g., extreme latencies in networking. This paper summarizes recent applications of EVT in networking from 2022 to 2023 to provide an overview of the current state of the field. The selected nine papers cover the application areas of Flow-Level Tail Latency, Ultra-Reliable Low Latency Communication, Dynamic Service Chaining, Mobile Edge Computing and Root Cause Location. EVT is a powerful method to improve various services, especially for those who need ultra-reliability. However, EVT has limitations. The biggest ones are that the quality of EVT depends on the data volume used, the confidence level employed in the distribution fitting, and the approach used for return level calculation.**

***Index Terms*—extreme value theory, recent applications in networking, literature review, survey**

1. Introduction

Extreme Value Theory (EVT) is a statistical method typically used to predict extreme events and model the data's tail behavior. Its application areas are wide-ranging, from natural catastrophes to engineering. Recent applications, especially in networking, are of interest due to their considerable potential for utilization. Therefore, this paper summarizes the most relevant papers from 2022 to 2023. In order to estimate which works are relevant, the citation and viewing numbers, as well as the standing of the publisher, were taken into account. Papers that do not actively incorporate EVT in their methodology, such as those employing it merely for validating their proposed systems (e.g., Chaccour et al. [1]) or relying solely on an EVT-based model (e.g., Pan et al. [2]), are excluded from consideration. This paper first introduces EVT to establish a foundational understanding and then presents the summaries of the selected papers. The papers cover the application areas of Flow-Level Tail Latency (Chapter 3.1.), Ultra-Reliable Low Latency Communication (Chapter 3.2.), Dynamic Service Chaining (Chapter 3.3.), Mobile Edge Computing (Chapter 3.4.), and Root Cause Location (Chapter 3.5.). Some application areas summarize more than one paper. In the end, the paper concludes with its own take on the field.

2. Understanding Extreme Value Theory

As described by Coles in [3], EVT has grown into an essential statistical model for applied sciences over the last few years. EVT models the tail distribution of empirically collected data and can even be used to predict future extreme events. The characteristic feature of extreme value analysis aims to quantify the stochastic behavior of a process at extremely large or small levels. The extreme value analysis typically requires an estimate of the probability of extreme events that surpass the already observed events. The following subsections present two commonly used distribution approaches of EVT.

2.1. Generalized Extreme Value Distribution

The Generalized Extreme Value (GEV) distribution combines the distribution families of Gumbel, Fréchet and Weibull. Using the combination is more effective than computing which distribution fits the dataset the best. The combined distribution is shown in Equation (1). [3]

$$G(z) = \exp\left\{-\left[1+\epsilon\left(\frac{z-\mu}{\sigma}\right)^{\frac{-1}{\epsilon}}\right]\right\} \qquad (1)$$

Equation (1) has three parameters. μ describes the location, σ the scale and ϵ the tail. By analyzing ϵ through inference, the data autonomously identifies the most suitable tail behavior, eliminating the need for subjective a priori judgments regarding the adoption of a specific extreme value family. Additionally, the uncertainty in the inferred value of ϵ quantifies the lack of certainty about which of the original three distribution families is most appropriate for a given dataset. GEV is used to model the distribution of block maxima. It separates the data into blocks of the same length and fits the GEV to the resulting set of block maxima. The choice of block size becomes critical when applying this model to any dataset. This decision involves balancing bias and variance. Smaller blocks may lead to poor model approximation, while larger blocks increase estimation variance. There are different methods to estimate the parameters of GEV. The most common is likelihood-based. However, by employing the likelihood-based method, a challenge arises around the regularity condition. This condition is essential for ensuring the validity of typical asymptotic properties linked to the maximum likelihood estimator. This challenge emerges from the GEV model because the endpoints of the distribution are functions of the parameter values. In the next subsection, another EVT approach solves this problem. [3]

2.2. Generalized Pareto Distribution

Focusing solely on modeling block maxima is an inefficient approach to extreme value analysis when additional

 doi: 10.2313/NET-2024-04-1_11

data on extremes is accessible. [3]

As described by Haan et al. [4], the Generalized Pareto Distribution (GPD) uses the Peaks over Threshold (PoT) approach instead. PoT categorizes all data points surpassing a chosen threshold as part of the tail. Equation (2) and (3) describe the GPD approach.

$$H(z) = 1 - \left(1 + \frac{\epsilon z}{x}\right)^{\frac{-1}{\epsilon}} \tag{2}$$

$$x = \sigma + \epsilon\,(y - \mu) \tag{3}$$

GPD has the same three parameters as GEV. y is the selected threshold. Altering the block size, even if it remains large, would impact the GEV parameters but not GPD. ϵ remains constant to block size changes, and the computation of x in Eq. (3) also remains unaffected. Variations in μ and σ are self-compensating. The GPD distribution, once fitted, has various applications. One possibility is to compute the return level corresponding to a given return period. This calculated value represents the extreme event. On average, an extreme event occurs once during that period. [3]

3. Recent Applications in Networking

The following subsections summarize nine different papers. Each summary consists of, when deemed necessary, a brief introduction to the topic, followed by an exposition of the proposed contributions of the discussed paper. Then, this paper explains the methodology used to make these contributions and presents the research results. All nine papers apply the EVT as shown in Figure 1. The papers first collect and filter data and then assess whether the collected data is suitable for the application of the EVT. Subsequent subsections discuss the criteria for applying the EVT. The papers gather additional data if the criteria are not met. On the other hand, if they are satisfied, the parameters of the EVT (such as the threshold value for the GPD approach) are calculated. Finally, the papers validate and apply their EVT model.

3.1. Flow-Level Tail Latency

This subsection summarizes the paper of Helm et al. [5].

Requirements at the end-to-end latency can be used in service-level agreements for communication networks and can, therefore, influence network planning and flow admission. These latencies can be measured and used as input for models, like EVT, to predict extreme latency occurings. The paper uses the PoT approach for 100 networks with random topologies, flow specifics and configurations to show that EVT can be applied to large datasets. The authors use 14 billion latency and jitter values from the measurements of Wiedner et al. [6]. Then, the EVT model is derived from the first 5 % of the data and validated on the remaining 95 %. Flow-level models outperform network-level models for high percentiles, suggesting that EVT models are more suitable at the flow-level when focusing on high percentiles of the tail. In addition, these models have a lower relative error of percentile values compared to network-level models. This result indicates their superior suitability despite having

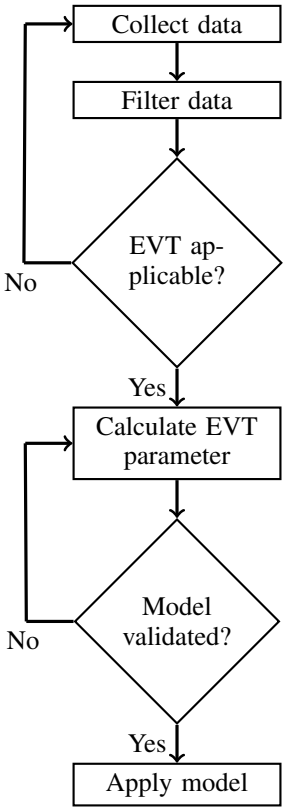


Figure 1: Overview flow of how to use EVT

less data. The accuracy of predictions ranges from 75 % to 85 % for twenty- and twofold time horizons. Two times the time refers to a duration corresponding to two times of the given horizon. The calculation is similar for twenty times. The paper forecasts tail latency quantiles at the flow-level with median absolute percentage errors between 0.7 % to 16.8 %. However, there are limitations to EVT. It depends on the volume of data, the confidence level of the distribution fitting, and the return level calculation. In addition, EVT is only applicable if the data is identically distributed and stationary. The authors use the Augmented Dickey-Fuller (ADF) test to ensure stationary. While in their setup, most flow latencies are stationary, this is not a general assumption.

3.2. Ultra-Reliable Low Latency Communications

This subsection summarizes the four papers from Mehrnia et al. [7]–[10]. These papers build on each other and use previous proofed results.

Ultra-Reliable Low Latency Communications (URLLC) is vital for 5th generation communication networks. An accurate channel modeling is needed since URLLC has a strict packet error rate and latency requirements. The paper [7] introduces a wireless channel modeling methodology based on EVT. The methodology involves deriving the parameters of the tail distribution by fitting the GPD to independent and identically distributed (i.i.d.) samples. To obtain these samples, the authors use declustering methods, Auto-Regressive Integrated Moving Average, and Generalized

Auto-Regressive Conditional Heteroskedasticity. After applying EVT, the mean residual life and parameter stability methods determine the optimum threshold. Next, the algorithm Minimum Sample Size Determination specifies the stopping condition to calculate the minimum required number of samples. Lastly, probability plots such as Probability/Probability (PP) and Quantile/Quantile (QQ) validate the channel tail model. The proposed framework requires significantly fewer samples than the conventional extrapolation-based approach. In addition, it fits the empirical data in the lower tail better.

The authors in [8] introduce a framework based on EVT to compute the optimal transmission rate in Ultra-Reliable Communication (URC). The authors consider a URC system that encounters fading, leading to diminished received power values. The system consists of a transmitter and receiver, sending packets over an unknown stationary channel. First, GPD represents the channel. The received power samples convert through declustering to i.i.d samples. These samples fit the GPD to the lower tail, and PP and QQ plots validate the Pareto model. Next, the optimal transmission rate is estimated. The function that selects the rate uses Pareto parameters. Finally, an evaluation of the error probability verifies the selected rate. The proposed framework outperforms traditional methods regarding reliability. Traditional methods use average statistic channel models. Moreover, because of the usage of the GPD threshold, the number of samples needed to achieve a certain reliability could be minimized.

In [9], a novel EVT-based framework is proposed to estimate the optimal transmission rate as well as the confidence interval on a small number of samples. The system model consists of one transmitter and one receiver communicating over a channel. The ADF test checks if the channel is stationary. If not, all factors that cause time variation of the GPD parameters are determined and collected as a sequence. This sequence is split into as many groups as needed, making each group stationary. The fixed transmission power is known in advance. First, the transmitter sends a packet to the receiver over an unknown channel. Then, the tail distribution is estimated by applying a modeling methodology based on EVT on the channel. Therefore, the statistics fit GPD to obtain power values surpassing the provided threshold. Moreover, the confidence intervals of wrong conclusions are derived for various numbers of samples. The intervals correspond to different probabilities. Lastly, the paper assesses the transmission rate by using EVT. Thus, the intervals of the Pareto parameters from different sample sizes are incorporated to achieve the desired error probability in URC. The paper validates its proposed framework with data collected in different sizes in a car engine. The targeted error probability is even met with limited data known.

As described in [10], the statistical method Multivariate Extreme Value Theory (MEVT) models the relation of rare events based on multidimensional limiting relations. MEVT is a further development of EVT and has additional functions like modeling dependence structures and joint distribution of several extreme events. The authors of [10] base their proposed channel modeling methodology on MEVT. The channel is for systems using Multiple Input Multiple Output (MIMO)-URC for efficiently deriving the lower tail statistic in multiple dimensions. The received signal powers are the data for these statistics. To validate the proposed methodology, the paper focuses on the bivariante or two-dimensional case. Before MEVT can be applied, the collected data converts into a sequence of i.i.d samples. Therefore, the above-introduced modeling of paper [7] is applied. Next, MEVT fits the GPD to the tail distribution to find optimal thresholds. Afterward, the Fréchet transformation is applied to each data sequence. Then, between the Fréchet sequences, the dependency factor is estimated. Next, two approaches are employed to fit Bi-Variate GPD (BGPD) to the joint distribution. The approaches used are the logistical distribution and the Poisson point process. Lastly, a mean constraint assessment validates the fitted BGPD model. The methodology is tested using one transmitter and two receivers in a car engine and compared to conventional models based on extrapolation. The proposed method performs significantly better in accurately modeling multiple dimensions events in URC.

3.3. Dynamic Service Chaining

This subsection summarizes the paper of Qin et al. [11].

Physical Machines (PMs) host Virtual Machines (VMs). VMs run software-based Virtual Network Functions (VNFs), which are enabled by Network Function Virtualizations (NFVs). The most essential requirement for service function chaining is guaranteeing ultra-reliable services. The existing research concentrates on average inter-failure time and repair downtime to define the reliability of VNFs. Due to uncertain PM failures, this does not fully capture the stochastic nature of VNF failure. The paper proposes a Dynamic Service Chaining (DSC) framework to examine the high-order statistics and probability of VNF failure time threshold deviation. The GPD approach of EVT characterizes the threshold deviation statistics with a low occurrence probability. The Poisson-Bernstein de la Harpe (PBdH) theorem describes extreme cases of PM failure time. A two-timescale VNF framework for mapping/remapping handles uncertain PM failure. The primary remapping framework works at a large timescale using matching theory. The optimal backup VNF framework operates at a smaller timescale. The algorithm used to find this backup effectively reduces computational complexity and balances switching costs and reliability. In addition, the backup needs to be selected beforehand. Simulation of the proposed DSC validates the PBdH. The randomly generated network topology is based on 20 nodes and 40 links. Other parameters are normalized. The numerical results show that using EVT to characterize extreme events improves service reliability compared to average-based schemes.

3.4. Mobile Edge Computing

This subsection summarizes first the paper of Liu et al. [12] and then of Ji et al. [13].

Traditional cloud computing has its resources pooled centrally. Mobile Edge Computing (MEC) has an advantage compared to traditional approaches because it

provides computing services close to the server. The authors address the challenges of offloading mission-critical tasks in MEC networks with Non-Orthogonal Multiple Access (NOMA). The network of the paper consists of a server and two sensor nodes, which supply the server with data. The server computes latency-sensitive tasks and works after the first-come, first-served principle. The overall error probability is characterized by the derivation of the Finite Blocklength (FBL) communication reliability and latency violation error probability through the GPD approach of EVT. The framework minimizes errors by jointly allocating the communication phase, the computation phase, and the user transmits power within stringent delay and energy constraints. The modified Block Coordinate Descent method addresses the non-convex problem by optimizing the time duration or proving the problem by characterizing the joint convexity of FBL error probability. Numerical simulations confirm the near-optimal performance of the proposed approach. Moreover, the paper's proposed framework outperforms the NOMA scheme with infinite blocklength solutions and the time-division multiple access scheme.

The authors in [13] address the issue of energy-efficient computation offloading in MEC systems on mobile applications with sequential or parallel module dependencies. First, the authors model mobile applications as Directed Acyclic Graphs. By considering the parent and children set of each computation module, the execution dependency gets handled. Then, the GEV approach of EVT is applied to explicitly address uncertainties and limit the occurrence probability of extreme events. Afterward, a newly developed ϵ-bounded algorithm, based on the column generation technique and with theoretical optimality guarantees, solves the offloading problem energy-efficiently. ϵ is the tail of the GEV model, and the optimal offloading policy is when ϵ is 0. Tools like Smart Diagnoses, tPacketCapture, WiFi SNR and PETrA were used to measure and record statistics. The result is a computation scheme outperforming other state-of-the-art schemes, such as Hermes and JSCO, in experiments conducted on an Android platform. The proposed scheme consistently has the lowest energy consumption as long as ϵ is smaller than 0.05. When this happens, the local device can save up to $50\,\%$ of energy. The cause for this is that JSCO neglects to account for uncertainties inherent in dynamic radio channels with queueing delays. In contrast, Hermes introduces additional energy consumption attributed to communication overhead resulting from the continuous probing of the channel.

3.5. Root Cause Location

This subsection summarizes the paper of Yang et al. [14].

The increasing complexity of online services can lead to significant losses when abnormalities occur. The root cause location is vital to guarantee the stable operation of online services. Therefore, the paper proposes a location method based on Prophet and Kernel Density Estimation (ProphetKdeRCL). ProphetKdeRCL consists of two stages. The first stage is the abnormal detection of performance indicators. This stage introduces the Prophet Mutation Point Updating (PMPU) algorithm. The Prophet model fits trend items better, and the timing anomaly detection gets more accurate through the usage of an improved version of EVT. PMPU solves the problems of existing methods since it can detect irregularities in the lowest range. The second stage locates the root cause of abnormal indicators and uses two algorithms. One is an anomaly degree measurement algorithm based on a Kernel Density Estimation. The second one is a time window-based causality analysis algorithm. This algorithm analyzes latency dependency via an intermediate structure and a time window. The effectiveness of the proposed algorithm is validated through testing and evaluations of the public time series, the microservice application system fault detection, and root cause location datasets.

4. Conclusion

Extreme Value Theory is a robust statistical method for predicting occurring extreme events and tail behavior modeling. The focus of this survey paper is on the most recent applications in networking from 2022 to 2023. This paper excludes papers that only build on EVT-based models or EVT for validating the paper's proposed methodology. First, this paper establishes an understanding of EVT. Therefore, it explains the distributions of Generalized Extreme Value and the Generalized Pareto Distribution in detail. Second, the nine selected papers are summarized. Each summary consists of the contribution of the paper, used methodologies, and key findings. Table 1 gives an overview of all discussed papers. Each line represents one of the nine papers. The order is the same as the papers are summarized in this paper. The Approach column shows that EVT's GPD approach is preferred over the GEV approach. All papers validate their EVT-based approaches and evaluate that their approach is superior to traditional ones in the discussed scenarios. They prove that accurate extreme event modeling and improving reliability predictions are possible. Therefore, they test their approach through virtual simulations or in the real world. None of the papers published their data to replicate their tests, and none have a Reproducibility Badge from the Association for Computing Machinery. Nevertheless, EVT has limitations, but only [5] highlights them. The quality of the EVT depends on factors such as the data volume, the confidence level employed in the distribution fitting, and the approach to return level calculation. Another weakness of EVT is that it can only be applied if the data is identically distributed and stationary. The data and used communication channels have to be either tested for stationary or assumed to be stationary, which leads to more calculation effort and complex systems. In conclusion, it can be said that EVT is a powerful tool to model and predict extreme values if the right approach is selected, the data is optimally fitted, and enough meaningful data volume is available. If not, EVT increases the system complexity and delivers wrong predictions. EVT is especially useful in the field of telecommunication since the demand for a method that can handle computation-intensive and latency-critical tasks is met.

TABLE 1: Overview table of all summarized papers

Paper	Approach	Validated	Tested
Helm et al. [5]	GPD	Yes	Virtual
Mehrnia et al. [7]	GPD	Yes	Real-World
Mehrnia et al. [8]	GPD	Yes	Real-World
Mehrnia et al. [9]	GPD	Yes	Real-World
Mehrnia et al. [10]	BGPD	Yes	Real-World
Qin et al. [11]	GPD	Yes	Virtual
Liu et al. [12]	GPD	Yes	Virtual
Ji et al. [13]	GEV	Yes	Real-World
Yang et al. [14]	improved EVT	Yes	Virtual

References

[1] C. Chaccour, M. N. Soorki, W. Saad, M. Bennis, and P. Popovski, "Can terahertz provide high-rate reliable low-latency communications for wireless vr?" *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9712–9729, 2022.

[2] C. Pan, Z. Wang, H. Liao, Z. Zhou, X. Wang, M. Tariq, and S. Al-Otaibi, "Asynchronous federated deep reinforcement learning-based urllc-aware computation offloading in space-assisted vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 7377–7389, 2023.

[3] S. Coles, *An Introduction to Statistical Modeling of Extreme Values*. Springer London, 2001.

[4] F. Haan, *Extreme Value Theory*. Springer New York, NY, 2010.

[5] M. Helm, F. Wiedner, and G. Carle, "Flow-level tail latency estimation and verification based on extreme value theory," in *2022 18th International Conference on Network and Service Management (CNSM)*, 2022, pp. 359–363.

[6] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "Hvnet: Hardware-assisted virtual networking on a single physical host," 2022.

[7] N. Mehrnia and S. Coleri, "Wireless channel modeling based on extreme value theory for ultra-reliable communications," *IEEE Transactions on Wireless Communications*, vol. 21, no. 2, pp. 1064–1076, 2022.

[8] ——, "Extreme value theory based rate selection for ultra-reliable communications," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 6, pp. 6727–6731, 2022.

[9] ——, "Incorporation of confidence interval into rate selection based on the extreme value theory for ultra-reliable communications," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2022, pp. 118–123.

[10] ——, "Multivariate extreme value theory based channel modeling for ultra-reliable communications," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2023.

[11] S. Qin, M. Liu, and G. Feng, "Dynamic service chaining for ultra-reliable services in softwarized networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3585–3595, 2023.

[12] Z. Liu, Y. Zhu, Y. Hu, P. Sun, and A. Schmeink, "Reliability-oriented design framework in noma-assisted mobile edge computing," *IEEE Access*, vol. 10, pp. 103 598–103 609, 2022.

[13] T. Ji, C. Luo, L. Yu, Q. Wang, S. Chen, A. Thapa, and P. Li, "Energy-efficient computation offloading in mobile edge computing systems with uncertainties," *IEEE Transactions on Wireless Communications*, vol. 21, no. 8, pp. 5717–5729, 2022.

[14] Y. Yang, Y. Sun, Y. Long, J. Mei, and P. Yu, "Root cause location based on prophet and kernel density estimation," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 904–917, 2023.

Network Applications of Trusted Execution Environments

Tim Kruse, Florian Wiedner*, Marcel Kempf*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: tim.kruse@tum.de, wiedner@net.in.tum.de, kempfm@net.in.tum.de

***Abstract*—As the interest in security in the networking community steadily increases, so does the interest in applying Trusted Execution Environments (TEE). However, despite the increased usage of TEEs, there is little information on how they are actually employed. To shed more light on an important tool for securing networking and its applications, we will present usages of the technology with a focus on networking. We found several proposals to utilize TEEs for networking applications in research, such as TrustedGateway, where the entire traffic is routed through the TEE. There are usages that enhance privacy by encrypting and decrypting data directly in the TEE. Finally, there are uses in cloud computing. We find that the usage of TEEs in networking applications is not that common at this time but seems to be a topic of active research.**

***Index Terms*—trusted execution environment, cloud computing, networking**

1. Introduction

Security in software products is of increasing importance, with bad actors aiming to gain access to data. There are many approaches to fortify different services and programs running on the network. One of these approaches is to employ Trusted Execution Environments (TEE), which aim to separate selected processes from the other parts of the machine. The processes are isolated from the system, which ensures that neither their code and data nor their execution can be compromised. Since the world is very connected, we want to survey the usage of TEEs to find how widely and for what purpose they are used in scientific projects in networking or networked applications. We will also determine whether there are any approaches that use TEEs in such a context in the field. For these approaches, we will show what the impact of the inclusion of a TEE is, with a focus on networking aspects, such as the impact on performance, e.g., latency or bandwidth, that can be expected. In this paper, we will first give a brief explanation of TEE, which will be followed by a summary of some interesting and diverse applications of TEEs in the space of networking and distributed systems. One interesting approach is to utilize a TEE to secure a gateway, which is called **TrustedGateway**. In this approach, all packets are passed through the TEE to applications in the cloud space. Other approaches use TEEs as a privacy-preserving measure. Since TEEs are present in most architectures these days, as Intel, AMD, and ARM have implemented them in hardware, their use will most likely continue to increase as the formation of the **Confidential Computing Consortium (CCC)** seems to indicate. The three manufacturers, as well as large companies in the cloud space, such as Microsoft and Google, are a part of this organization. We will first start with an outlook on related work in Section 2, which will give an overview of what TEEs are and also provide some information on the CCC. This will be followed by Section 3, where we will showcase applications that we found, such as TrustedGateway, which runs all traffic through a TEE. We will also take a look at applications that utilize a TEE to provide privacy. We will give information about the current usage of TEEs in cloud computing. In Section 4, a summary of the usages we found will be given, alongside our opinion on which topics warrant further research interest.

2. Related Work

In this section, we will present key components that will be relevant to this survey in order to enable the reader to follow the applications and reasons for their usage. Furthermore, we provide a starting point to conduct further research into the field.

2.1. Trusted Execution Environment

A TEE is a tamper-resistant software environment that is part of the processor. It aims to provide trust that a piece of code is executed as it should. According to Sabt et al. [1], it relies on a chain of trust that is established during the boot process in order to ensure that the environment can establish the authenticity and confidentiality of the code executed within it. Furthermore, the TEE provides integrity protection and an attestation mechanism to provide proof for the execution. Both the TEE and the rich environment, which houses the operating system, run on a separation kernel to isolate them from each other. To establish the TEE, Sabt et al. [1] propose the following five key building blocks, as shown in Figure 1:

- **Secure Boot** Ensures that if code is modified, it is detected and the chain of trust is seen as broken, provides a **Trusted Computing Base (TCB)**, which encompasses all security-critical hardware and software of a system
- **Secure Scheduling** Provides Scheduling to ensure that the rich environment is responsive
- **Inter-Environment Communication** Exchanges data between the rich environment and the TEE

 doi: 10.2313/NET-2024-04-1_12

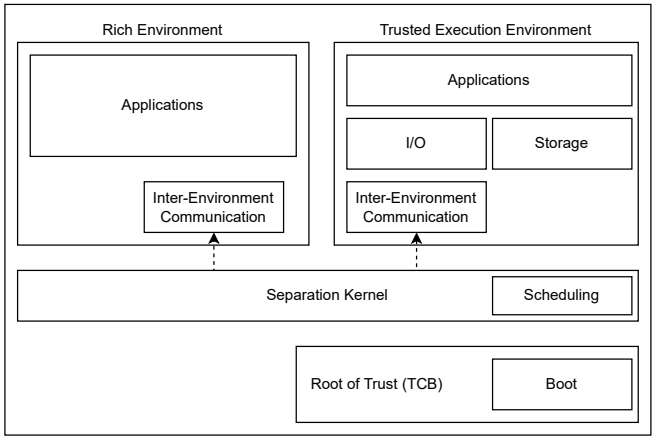


Figure 1: Building Blocks of a TEE, adapted from [1]

- **Secure Storage** Utilizes authenticated encryption, can only be used by the TEE
- **Trusted I/O Path** Provides trust for the peripherals to ensure that their input is not changed

There are many different TEEs available at this time, but the most important ones are Intel SGX [2], and its extension Intel TDX [3], AMD's SEV [4], and ARM's TrustZone [5]. The first two are mostly found in devices such as servers and PCs, while the ARM implementation is found in mobile phones and commodity gateways, among others [6]. These products are in continuous development as attack vectors are found over time. This led to the extension of Intel SGX with Intel TDX. The products are dependent on specific hardware, which leads to issues with the portability of code, as the interfaces differ between vendors. This resulted in efforts to standardize these interfaces [7], though different approaches, such as Open-TEE [8], which offers a virtual TEE to develop code and compile it for different TEEs, were also developed. TEEs aim to shield from all software and hardware attacks, but there are some attack vectors that exist despite these efforts, such as side-channel attacks, as shown by Wang et al. [9]. Despite that, the number of applications for TEEs seems to be ever-increasing, as research by Geppert et al. [10] shows. They tried to ascertain use cases and challenges for the use of TEEs, with a focus on cloud computing use cases.

2.2. Confidential Computing

In 2019, the Confidential Computing Consortium was created to advance the security of user data in the cloud [11], to describe different means to ensure the security and privacy of programs and data by utilizing TEE, as outlined in their report [12]. One of their aims is to provide secure and private ways to allow multiple physical nodes to compute tasks, as is common in the cloud space, for which they use TEEs to provide confidentiality as well as integrity. The consortium consists of large companies in the space. Key among them are the big hardware vendors Intel, AMD, and ARM, together with cloud operators such as Microsoft and Google [13]. They have outlined multiple use cases in their paper [12], which they want to employ to improve the security and privacy of "data in use" [12]. They aim to utilize the advantages of TEEs not only for normal cloud operations but also to facilitate the increased use of multi-party computation. This is aimed at data that can not be given to another party due to privacy concerns. In this setup, computation can still be done efficiently without compromising the data's security and privacy.

3. Applications of TEE in Networking

In the following section, we will give an overview of some of the uses of TEEs in a networking context.

3.1. TrustedGateway

One application by Schwarz [6] aims to increase the security of gateways. They deem the security of commodity gateways under threat, since most gateways have an increasing number of additional, less secure services running that offer services like FTP or a VPN into the network guarded by the gateway. Since it is possible to compromise the gateway via these additional services, these attack vectors could be used to compromise the core tasks of the gateway, as described in their paper. As a solution to this problem, they move all network traffic through a TEE. To achieve their goal, they implemented the minimum needed for securely networking traffic through the ARM TrustZone TEE. In order to keep performance as good as possible and limit the amount of TCB required, they only implemented switching, routing, and the firewall into the TEE.

3.1.1. Design of TrustedGateway. Schwarz [6] proposes two parts to realize their goal. The first is a networking utility called **NetTrug**, which performs the networking tasks. The second is called **ConfigService**, which is used to configure NetTrug. Since we are primarily interested in networking and not secure policy configuration, we will focus on the NetTrug utility. The entire project is implemented using **Open-TEE (OP-TEE)**. It is composed of two major parts, a partial networking driver we trust and its untrusted counterpart running in the rich environment. The second part is the I/O workers, which remove the need for context switches by handling network interrupts. NetTrug is the only application of the system that has direct access to the network interface cards (NIC), which are then considered trusted interfaces. There can be untrusted interfaces, but these will still have to run through NetTrug. The traffic is filtered by a stateful L3/L4 firewall, in their case NPF [14], but the utility itself has no complete protocol stack. To make the filtering transparent for the applications running in the rich environment, a virtual network interface card (VNIC), which is based on **virtio-net** and **virtio-mmio**, with input and output queues, is used to connect NetTrug to the rich environment. To prevent spoofing from both a system or outside attacker, the following mechanism is in place: The IP addresses for all NICs are static. Whenever a packet is sent or received, the MAC address of the NIC is used. This limits inbound traffic as well, as only traffic sent to one of the IP addresses of the TrustedGateway is accepted. The network stack deployed in the TEE is minimal. It offers Address Resolution Protocol (ARP) resolution for itself and also checks the traffic for ARP spoofing attacks originating

 doi: 10.2313/NET-2024-04-1_12

from both the rich environment and an outside attacker. It also replaces the MAC for the VNIC, which it swaps into the destination spot when forwarding the traffic. Since NetTrug does not offer an IP or UDP stack, to keep the TCB small, it also does not offer DHCP or DNS services, and the NIC's IP addresses are statically assigned. The security of the configuration of NetTrug must also be ensured, as otherwise, the entire effort would be for naught. This is done by the second part of the TrustedGateway, the ConfigService. It is an OP-TEE application, which is binary-signed, integrity-checked, and protected against version rollbacks. In order to protect it against attackers, it only allows trusted devices to configure it, which is done by the HTTPS client in the TEE. This client only performs the TLS portion, while the TCP portion is performed in the rich environment. It also utilizes custom request headers to avoid cross-site request forgery attacks. There is a master certificate that is uploaded at a physically attached interface when first configuring the service, and the service itself creates one when it is first started. The master admin is the only entity that can add or remove further admin certificates. Within this utility, it is possible to define rules for the firewall or assign different IP addresses for the NICs.

3.1.2. Performance of TrustedGateway. Schwarz poses in their paper [6] that due to TrustedGateway extracting only the strictly necessary services into the trusted environment, the TCB required is reduced compared to a full stack in an operating system, as it offers less functionality. This, in turn, reduces the likelihood of vulnerabilities. Furthermore, the performance impact of the setup on a TCP connection was measured. To have a comparable baseline setup for experimentation, they disabled NIC offloading, as it is not implemented in TrustedGateway. They set up three different experiments using iPerf3 [15], with the first two running a receiver (1) and a sender (2) in the gateway's rich environment, respectively. The third experiment was set up such that the gateway had to forward the traffic to a host outside the network connected to the gateway or receive traffic through the gateway. The key performance metrics they observed were the network throughput, the network latency, and the overhead of the firewall inside the TEE. In experiment (1), they noticed that the throughput was roughly 90% of the normal throughput with 385 Mbit/s, with experiment (3) delivering roughly the same results. In experiment (2), in some cases, higher throughput than the baseline of 369 Mbit/s could be observed. In experiment (3), the performance with the sender inside the network was at around 93% of the baseline, which equates to 236 Mbit/s, and when receiving, reached between 101.9% - 103.5% of the baseline with 221 Mbit/s. The firewall overhead was also observed in the same experiment setup and was in the range of 0.5% to 1%, though the baseline value is not mentioned, besides the result being called small. The author attributes it to NPF's static code. Another very important metric was latency overhead, as it is critical for low-latency networked applications, such as phone calls. To test the latency in a user-like manner, they loaded different websites from the Tranco List [16], which is a list of websites that should be used in research, as they are aimed to be hardened against manipulation. The latency overhead was around 3.4% on average, peaking at 4.95%, though no baseline latency is given. The latency is attributed to the TrustedGateway workers having to be woken up, which the author tried to avoid by employing an idle grace period. To evaluate low latency performance, they sent ping packets from a host inside the network to an outside host via the gateway. In this case, the average overhead was even lower, with around 2.7%, which equates to 0.37ms of added latency when utilizing the TrustedGateway compared to the stock setup. The config service is also responsive despite the fact that it runs in both the rich and trusted execution environment, with a load time of around 1-2 seconds. Since the TCB is rather small, with around 110k lines of code (LOC), rather than the 4523k LOC included in the router's default operating system, including both the OP-TEE and its cryptography library, the memory it needs is fairly small, with 32 MB of which 20 MB are for trusted user apps, which could offer other functionalities. This small code base should increase security, as larger code bases are more susceptible to issues simply due to their size. Overall, the usage of a TEE for this purpose seems sensible, especially given the fact that dedicated gateways are expensive, and this allows for secure operation of the main purpose, namely forwarding and protecting the network behind it.

3.2. TEE for Privacy

Since TEEs can provide confidentiality for the data that they use as well as prove the integrity of their computation, they are very useful for maintaining privacy. Multiple proposals in the networking space, therefore, utilize TEEs to provide privacy. We are going to focus on two specific applications that show the breadth that TEEs can be used for. In the first section, we are going to present a TEE utilized by Risdianto et al. [17] to deploy traffic policies across organizations. The second section focuses on the use to provide secure communication between Android devices which was proposed by Wang et al. [18].

3.2.1. TEE-based Collaborative Traffic Policy. Since traffic between a company's different sites tends to be forwarded via the same routers and firewalls, there is a possibility for policies to erroneously direct traffic via a public link rather than a preferred private link. In order to alleviate this problem and also allow different companies to collaborate in a manner that does not require them to exchange their policies, Risdianto et al. [17] propose the use of a TEE-based approach. They aim to use it for programmable network switches, as are in use at many larger commercial operations. They propose a way to compile their policies such that the result can be combined with an organization's own rules. Both parties need to input their data into the enclave, which refers to SGX's per-program TEE space, and then the data can be exchanged via an SSL connection. Each connection is only usable once and unidirectional, meaning that party A's enclave transfers its policy to B's enclave, and B's transfers it to A's. Both sides can then compile their policies into rules. During this phase, there are also two steps to check for overlaps between the rules. The first part is the inter-policy check, which checks for exact matches between the rules. The second portion is an intra-policy check, which employs

a binary trie to find overlapping addresses and discards addresses such as 0.0.0.0/0 as they can lead to leaks. The rules are then compiled such that they can be installed on a P4 switch, which is a software-defined network device that can be programmed to perform on all layers of the network stack using the language P4 [19].
The use of a TEE in this case is very interesting as it allows for secure multi-party computation, enabling a shared configuration without exposing your own policies to a less trusted party. According to Risdianto et al. [17], it does not have an impact on the performance of the routers and firewalls, as this process only aims at the compilation of rules rather than the actual operation. It also requires the use of P4 routers. Furthermore, it makes use of the remote attestation mechanism of SGX, as the policy compilation is done in the TEE, which can be used to prove that the same configuration was used by both parties. This has the major advantage of making it simpler to create a communication link between two organizations. It is, however, important to add that several parts of SGX have been proven to be vulnerable to attacks, even the remote attestation, as summarized by Fei et al. [20].

3.2.2. TEE-Based Communication on Android. In the second approach, Wang et al. [18] propose means to have a secure communication channel between Android devices, for which they leverage ARM's TEE called Trustzone. In order to secure communication between parties, each device uses the Elliptic-Curve Diffie-Hellman (ECDH) key agreement to create keys, which are stored inside the TEEs and never leave them, to ensure that they can not be exposed. The system is separated into two parts inside the TEE, namely the key sender and the key receiver. The key sender facilitates the public key generation and the generation of the digital signature that is used to provide integrity proof for the session key negotiation messages. The key receiver is responsible for the public key authentication, the session key generation, and key storage. Both entities run in OP-TEE OS and are involved in sending and receiving operations. Whenever a user wants to send something to another party, they first type the data in the rich environment, then send the data into the TEE to encrypt it using the appropriate keying material, which was negotiated before. Then, the encrypted message is sent to the other party, which can only decrypt the data in the TEE.
It is important to note that in their paper Wang et al. [18] did not actually implement the networked portion. Therefore, their performance analysis is not necessarily correct, but they did analyze the amount of time that their additional measures took and estimated the networking portion. The ECDH key agreement took around 0.563s, from which they assumed the transmission to take place with 128KB/s, which results in a total transmission time of 0.059s. However, this does not need to take place each time two parties want to communicate, as the keys can be kept for a longer time frame. Their method of handling encryption with a TEE also has an impact on performance compared to the standard Android Keystore technology [21], but with an impact of around 11%, it should be manageable as the agreement does not need to be performed each time the two parties communicate. Wang et al. [18] pose that the advantage of this approach is that there is no need for a trusted third party while still allowing two parties to ensure the integrity and confidentiality of their communication. The TEE also decreases the risk of the key being stolen, which can be a problem in Keystore, as it stores the keys in a file. Overall, the performance of an application is worse if a TEE is employed, as it requires additional communication within a system, but it provides more security.

3.3. Usage of TEE in the Cloud

Another interesting use of a TEE is one that seems at the center of the CCC: The usage of TEEs in cloud applications. Since "the cloud" refers to a large distributed system that handles computation to offload tasks from a local device, it is important that they can maintain the confidentiality of their data as well as authenticate that they processed it correctly. The three largest cloud providers, AWS [22], Azure [23], and Google Cloud [24], all offer services that give you access to a TEE. For both Microsoft and Google, who are part of the CCC, this is offered under confidential computing, with Microsoft offering customers access to VMs with Intel SGX and AMD SEV. However, the access to the TDX extension is limited at this point. Google, on the other hand, only offers SEV [25]. Both, however, also offer other types of confidential computing, which offer different pieces to ensure secure operation. Amazon offers its own service that is built on top of Intel and AMD processors, as well as its own architecture, Graviton. Intel TDX itself is also built on top of SGX but is meant to offer capabilities that enable cloud computation. Cheng et al. [26] have offered an overview of the TDX technology. Intel TDX is built for cloud computation, as it runs on top of Intel VT, their virtualization technology. In the work by Geppert et al. [10], they offer multiple use cases for TEEs in the cloud, such as the ability to move data from on-site into the cloud while maintaining protection. Another case is multi-party computation, where the parties need to be able to rely on each other to have computed correctly, which can be proven by the attestation mechanisms offered by TEEs.
Overall, the usefulness of TEEs in cloud applications seems very high, and they must be in use, as the three biggest cloud providers offer them to their customers. However, it is important to note that there is little actual data on the manner in which TEEs are actually used in cloud applications.

4. Conclusion

In this paper, we aimed to give a broad overview of how networking and networked applications are realized in the context of TEEs. We showed different avenues, from directly running the network traffic through the TEE in the case of TrustedGateway to using it as a means to enhance privacy from both a network administration perspective and to secure messages between two Android devices. We also gave an overview of the current state of TEEs in the cloud space, where we showed that they seem to be in demand as the major operators offer services but could not find concrete data on the subject. In summary, there is a lot of interest in the technology, though mostly as a means to increase privacy, for which TEEs are very well suited.

However, there is not a lot of information about how to create applications that use networking directly in the TEE without leveraging the rich environment to perform the Network I/O. Nonetheless, this area is interesting, and the TrustedGateway approach offers a good idea of how that could be done. Since it is a highly specific application, it remains to be seen if the approach can be used in other cases as well. In future research, it would be interesting to see if more applications take an approach similar to the one employed by TrustedGateway and how the usage of TEEs in cloud applications develops.

References

[1] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2015.

[2] Intel, "Intel® Software Guard Extensions (Intel® SGX)," 15/11/2023. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html

[3] ——, "Intel® Trust Domain Extensions (Intel® TDX)," 04/11/2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html

[4] AMD, "AMD Secure Encrypted Virtualization (SEV)," 15/11/2023. [Online]. Available: https://www.amd.com/en/developer/sev.html

[5] Arm Ltd., "TrustZone for Cortex-A – Arm®," 16/11/2023. [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-a

[6] F. Schwarz, "TrustedGateway: TEE-Assisted Routing and Firewall Enforcement Using ARM TrustZone," in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 56–71.

[7] GlobalPlatform, "Specifications Archive - GlobalPlatform," 15/11/2023. [Online]. Available: https://globalplatform.wpengine.com/specs-library/?filter-committee=tee&utm_source=iseepr&utm_medium=Website&utm_campaign=TEE

[8] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-TEE – An Open Virtual Trusted Execution Environment," in *2015 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2015.

[9] J. Wang, Y. Cheng, Q. Li, and Y. Jiang, "Interface-Based Side Channel Attack Against Intel SGX." [Online]. Available: https://arxiv.org/pdf/1811.05378.pdf

[10] T. Geppert, S. Deml, D. Sturzenegger, and N. Ebert, "Trusted execution environments: Applications and organizational challenges," *Frontiers in Computer Science*, vol. 4, p. 930741, 2022.

[11] Redmondmag, "Confidential Computing Consortium Formed To Protect Processed Data – Redmondmag.com," 05/12/2023. [Online]. Available: https://redmondmag.com/articles/2019/08/21/confidential-computing-consortium.aspx

[12] Confidential Computing Consortium, "Confidential computing: Hardware-based trusted execution for applications and data v1.3," 11/2022. [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf

[13] "Members – Confidential Computing Consortium," 15/11/2023. [Online]. Available: https://confidentialcomputing.io/about/members/

[14] GitHub, "rmind/npf: NPF: packet filter with stateful inspection, NAT, IP sets, etc," 05/12/2023. [Online]. Available: https://github.com/rmind/npf

[15] V. Gueant, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool," 26/02/2024. [Online]. Available: https://iperf.fr

[16] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019.

[17] A. C. Risdianto and E.-C. Chang, "TEE-Based Privacy-Preserve in Collaborative Traffic Policy Compilation for Programmable Devices," in *Proceedings of the 2021 ACM International Workshop on Software Defined Networks & Network Function Virtualization Security*, ser. SDN-NFV Sec'21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 19–22.

[18] Y. Wang, W. Gao, X. Hei, I. Mungwarama, and J. Ren, "Independent credible: Secure communication architecture of Android devices based on TrustZone," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, pp. 85–92.

[19] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[20] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security Vulnerabilities of SGX and Countermeasures," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, 2022.

[21] Google, "Android Enterprise Security Paper 2023," Google, 2023.

[22] Amazon Web Services, Inc., "Nitro Enclaves," 01/12/2023. [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[23] Microsoft, "Azure confidential computing products," 01/12/2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/overview-azure-products

[24] Google Cloud, "Confidential Computing | Google Cloud," 01/12/2023. [Online]. Available: https://cloud.google.com/confidential-computing

[25] ——, "Confidential Computing concepts," 01/12/2023. [Online]. Available: https://cloud.google.com/confidential-computing/confidential-vm/docs/about-cvm

[26] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX Demystified: A Top-Down Approach," 2023.

LoRaWAN – Current State, Challenges, and Chances

Benjamin Liertz, Jonas Andre, Leander Seidlitz*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: benjamin.liertz@tum.de, advisor@net.in.tum.de

***Abstract*—**

Currently used Internet of Things (IoT) devices mostly use WLAN, Bluetooth, and Zigbee, limiting the communication range and battery life. LoRaWAN is a new low-power, long-range wireless technology that enables battery-powered IoT devices with a lifetime of more than 10 years. This paper aims to give a good understanding of what LoRaWAN is and how it works. Particular attention is paid to performance in terms of flexibility and power efficiency. The modulation technology is also analyzed in more detail, which enables demodulation with a Signal to Noise Ratio (SNR) of less than -22 dB and a range of up to 15 km. Finally, the biggest problem of LoRaWAN, the scalability problem, is discussed, and current research is presented.

***Index Terms*—internet of things (iot), modulation, medium access control (mac), aloha, lora, lorawan**

1. Introduction

Most of today's IoT devices use WLAN, Bluetooth, and Zigbee, limiting the communication range to about 100m [1]. Another problem most devices share is their high power consumption, making battery-powered devices last only for a few years at best. Battery-powered long-range IoT devices would enable many new applications, especially in smart agriculture, smart cities, and industry 4.0.

Low Power Wide Area Networks (LPWAN) were introduced to address this issue. LoRaWAN is a relatively new LPWAN and stands for Long Range Wide Area Network [2]. Compared to short-range transmission standards, LoRaWAN is designed for wide-area coverage, low energy consumption, and cost-effective deployment of End Devices (EDs) [2], [3]. There are also other LPWANs like SigFox and Weightless. Still, LoRaWAN is often of greater interest because of its open business model and ability to constantly optimize time on air (ToA), energy consumption, and data rate. This reduces the deployment cost even further and creates a network that can adapt to the environment, while optimizing the battery life of the EDs [2], [4].

This paper presents a profound overview of LoRaWAN and its capabilities. Additionally, we will analyze LoRaWAN's performance regarding flexibility, energy efficiency, and scalability.

Section 2 gives an introduction to LoRaWAN and its network architecture. LoRaWAN's modulation technique is explained in Section 3. Section 4 describes the different device classes of LoRaWAN. In Section 5, the performance of LoRaWAN is analyzed. Finally, section 6 presents the scalability problem of LoRaWAN and current research.

2. Introduction to LoRaWAN

Two of LoRaWAN's key characteristics are high power efficiency and long transmission range. This enables battery-powered devices to last more than 10 years and a range of up to 15 km away from the next LoRa Gateway (GW) [1], [4]. Today, LoRaWAN is used worldwide in applications like reindeer tracking in Finland, smart fire alarms and fire detectors, smart bus schedule signs, and a city-wide network in Canada [5].

2.1. LoRaWAN vs. LoRa

LoRa and LoRaWAN are separate elements of the LoRa network, each associated with a different layer in the protocol stack [5].

LoRa, residing at the physical layer, is a wireless modulation technique that employs a variant of the Chirp Spread Spectrum modulation, providing the long-range communication link between the GWs and the EDs [1], [5].

LoRaWAN builds on top of LoRa. It defines a communication protocol and a system architecture for the LoRa network, specifying the Medium Access Control (MAC). As MAC, the ALOHA principle is used so EDs can initiate uplinks whenever they want, enabling the EDs to go into sleep mode the rest of the time to save power. The LoRa Alliance standardizes LoRaWAN, an open specification in contrast to the proprietary LoRa radio frequency modulation, which Semtech Corporation owns [2], [6], [7]. The protocol works in the unlicensed, worldwide, industrial, scientific, and medical (ISM) bands, so there is no licensing fee, but the devices have to follow the ISM rules [5]. The LoRaWAN protocol also manages routing, access control, and data encryption for EDs [7].

2.2. LoRaWAN Network

A LoRaWAN Network is a star-of-star network consisting of multiple network elements:

- The nodes of the star-of-stars topology are the LoRaWAN EDs, like temperature sensors or actuators such as valves. They are often battery-powered and use the LoRa radio frequency modulation to communicate with GWs [1], [5].

 doi: 10.2313/NET-2024-04-1_13

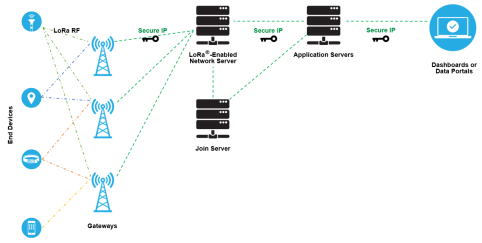


Figure 1: Typical LoRaWAN Network, Source: [8]

- GWs function as LoRa access points for EDs, forwarding all packages from the EDs to the Network Server (NS) to which they are connected via WLAN, Ethernet, or Cellular [5].
- At the core of every LoRaWAN network lies the NS. It is the hub of the star-of-star topology managing the entire network, from the connection between the GWs and EDs to the applications [1].
- The application server, lying in the backend, handles all the data generated by the EDs providing application-level services [1], [5].
- The Join Server manages the Over-the-Air activation process of adding ED to the LoRaWAN network [5].

3. LoRa Modulation and Demodulation

LoRa utilizes a patented modulation technique by Semtech known as Chirp Spread Spectrum. This technique encodes the data signal onto a chirp signal, a tone with a linearly increasing or decreasing frequency over time [1], [3]. The modulation technique spreads the modulated signal over a wide band beyond the original signal's bandwidth, making the signal less sensitive to selective frequency fluctuations [1], [2]. The modulation technique allows the demodulation of signals for an SNR even below -22dB, a link budget of up to 157dB, and a receiver sensitivity of just -137 dBm [1], [9], [10].

3.1. Modulation

For digital modulation schemes, a finite set of symbols is required. A flexible parameter of LoRa is the spreading factor (SF), which can be seen as the chirp rate and will be analyzed in section 5. LoRa utilizes 2^{SF} different symbols, with each symbol carrying 2^{SF} bits of information. In LoRa, a symbol is encoded with a cyclically shifted chirp, so the modulation technique is also called frequency shift chirp modulation [11].

The transmitted waveform $c(nT_s + kT)$ for a symbol $s(nT_s)$ in LoRa is defined as:

$$c(nT_s + kT) = \sqrt{\tfrac{1}{2^{\text{SF}}}} \cdot \exp\left(j2\pi \left[(s(nT_s) + k) \mod 2^{\text{SF}}\right] \tfrac{k}{2^{\text{SF}}}\right) \tag{1}$$

Here, $s(nT_s)$ represents the symbol number and k increments for each sample both take values in $\{0,1,...,2^{\text{SF}}-1\}$. $T = \frac{1}{BW}$ is the sample duration, and $T_s = 2^{\text{SF}} \cdot T$ is the symbol duration [11]. In the following, $s(nT_s) = q$ is used for the symbol to be transmitted.

The formula shows that q can be seen as the starting frequency of the waveform. The modulo operation ensures that when the chirp reaches the end of the bandwidth, it starts again at the beginning, rising until frequency q is reached again. So, different starting frequencies represent different symbols, as seen in Figure 2.

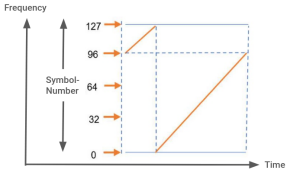


Figure 2: cyclically shifted up-chirp of symbol 96, Adapted from: [3]

3.2. Demodulation

Here, the challenge is in identifying the symbol that was transmitted. The idea is to compare the received signal $r(nT_s + kT)$ with the ideal symbols and pick the one with the highest correlation. Correlation is a mathematical way of measuring the similarity between two signals and is defined as the dot product of the two signals, described with the formula:

$$\sum_{k=0}^{2^{\text{SF}}-1} r(nT_s + kT) \sqrt{\tfrac{1}{2^{\text{SF}}}} \exp\left(-j2\pi \left((q+k) \mod 2^{\text{SF}}\right) \tfrac{k}{2^{\text{SF}}}\right) \tag{2}$$

So, to find the correct symbol for a received LoRa symbol, 2^{SF} similarity checks have to be performed. Each similarity check would need 2^{SF} multiplications and 2^{SF} additions, making it very inefficient, especially for higher SFs. To overcome this problem, we use a mathematical trick and add $+k-k$ to the exponential function, allowing us to separate the $+k$ term:

$$\begin{aligned} &\exp\left(-j2\pi \left((q+k) \mod 2^{\text{SF}} + k - k\right) \tfrac{k}{2^{\text{SF}}}\right) \\ &= \exp\left(-j2\pi \tfrac{k^2}{2^{\text{SF}}}\right) \cdot \exp\left(-j2\pi \left((q+k) \mod 2^{\text{SF}} - k\right) \tfrac{k}{2^{\text{SF}}}\right) \\ &= \exp\left(-j2\pi k^2 \tfrac{1}{2^{\text{SF}}}\right) \cdot \exp\left(-j2\pi qk \tfrac{1}{2^{\text{SF}}}\right) \end{aligned} \tag{3}$$

In Equation 3 the following observations can be made:

- $\exp\left(-j2\pi k^2 \frac{1}{2^{\text{SF}}}\right)$ does not depend on q anymore and is a simple down chirp.
- $\exp\left(-j2\pi qk \frac{1}{2^{\text{SF}}}\right)$ is a pure sinusoidal waveform where the frequency depends on q.

The correlation can now be rewritten as:

$$\textstyle\sum_{k=0}^{2^{\text{SF}}-1} r(nT_s + kT) \exp\left(-j2\pi \frac{k^2}{2^{\text{SF}}}\right) \sqrt{\frac{1}{2^{\text{SF}}}} \exp\left(-j2\pi qk \frac{1}{2^{\text{SF}}}\right) \tag{4}$$

Demodulation of the signal becomes:

1) Multiplying the received signal r with the down chirp of the base signal, converting the chirp into a single-frequency tone, representing the transmitted symbol.
2) Performing correlation with the bank of frequencies depending on q, This is functionally the same as performing the Fast Fourier Transform and picking the symbol with the highest peak.

This method of demodulating the signal is much more efficient than using the dot product. It's a key reason why LoRa is known for its power efficiency. [11], [12]

4. Device Classes

A problem with LPWANs is the conflict between lower power consumption and network downlink latency [5]. LoRaWAN addresses this problem by allowing the EDs to act according to one of the three classes: A, B, and C [2].

4.1. Class A

This device class includes all LoRaWAN EDs, as the name implies. So, every ED acts like a Class A device but can implement additional behavior to become a Class B or C device. Class A devices prioritize power efficiency and are, thereby, normally battery-powered.

Following an uplink transmission to the GW, Class A devices open two short downlink windows (RX1 and RX2) before entering a low-power sleep mode [1]. While Class A devices offer good power efficiency, downlink communications from the server must wait until the next uplink from the ED, introducing latency [5]. So this Class is optimal for low-power sensors focused on uplink, which are by now most of all LoRaWAN EDs and are thereby well studied [2].

4.2. Class B

Class B devices, also typically battery-powered, begin as Class A devices and negotiate with the network server to switch to Class B mode [2]. In addition to the uplink transmission and downlink windows RX1 and RX2, Class B devices open extra receive windows called ping slots at scheduled times without significantly increasing power consumption [1], [13]. To do so, the EDs and the GW synchronize their time via beacons broadcasted every 128 seconds from the GW to all EDs in range. This gives the NS opportunities to initiate a downlink, drastically reducing the downlink access delay compared to Class A and providing a perfect solution for actuators or sensors requiring command interventions [2].

To avoid systematic collisions and problems of overhearing messages between Class B EDs, the NS calculates a ping offset for each ED and every beacon period. It is a pseudo-random offset added to the start of the first ping slot in the beacon period, so Class B EDs have their ping slots at different times [2].

Class B devices themselves provide a trade-off between low downlink access delay and low packet loss with lower power consumption. The number of ping slots opened by the end of each beacon period is called the ping number and can be chosen from the ED [2]. A low ping number reduces lost packets and power consumption due to fewer collisions and more sleep time, perfect for ED in large LoRa networks prioritizing power efficiency. A higher ping number enables low access delay while increasing the power consumption, well-suited for ED needing lower latency. However, there is a tipping point for higher ping numbers where the access delay and packet loss ratio rises with the number of EDs due to more collisions caused by the increased network load [2].

Other advantages of Class B devices over Class A devices are that they significantly reduce packet loss of downlink traffic and can receive firmware over the air way more efficiently [2], [13].

Despite potential benefits, real-world implementations of Class B devices remain limited, because there is only an unmaintained buggy version available, necessitating further study and development [2].

4.3. Class C

Unlike A and B, Class C devices listen continuously, sacrificing power efficiency for constant availability. The device's receive windows remain open until the subsequent uplink transmission, ensuring uninterrupted communication availability. Therefore, the power consumption is relatively high, so Class C EDs are by default not battery-powered [1].

5. Performance

5.1. Time on Air

The ToA is the time it takes to transmit a message between the ED and the GW and is defined by the SF, the bandwidth, and the message size. A longer ToA results in higher power consumption because the ED has to transmit longer. Additionally, a higher ToA and the duty cycle limits of 1% in the ISM band result in a longer block duration, which is the time the ED has to wait until it can transmit again, which is especially critical for the GWs [7], [11], [14]. The MAC layer of LoRaWAN using ALOHA does not play well with a longer ToA because it increases the probability of collisions and retransmissions, resulting in a lower Packet Delivery Ratio (PDR) [2]. All in all, the ToA is a critical factor in the performance of LoRaWAN and should be as short as possible [7].

5.2. Spreading Factor

One parameter that appears all the time is the SF and, thereby, has a significant influence on the performance of LoRa. LoRa defines six orthogonal SFs, enabling simultaneous non-conflicting transmission on the same channel. The SF are defined as $\mathrm{SF} = \{7, 8, 9, 10, 11, 12\}$, resulting in 2^{SF} possible symbols [1], [2].

Increasing the SF by one also halves the rate at which the chirp changes its frequency, resulting in a halved data rate, as can be seen in Equation 1. A lower chirp rate additionally results in a higher receiver sensitivity and is less susceptible to noise and interference [15]. So, increasing the SF results in a higher receiver sensitivity, a higher ToA, and a lower data rate. Comparing SF7 and SF12, SF12 can typically still be demodulated with an SNR of -22 dB, while SF7 requires an SNR of -7 dB. However, SF12 has a ToA of 32 times the ToA of SF7 and a typical data rate of 0.3kbit/s compared to 5.5kbit/s of SF7 [9]. The choice of the SF enables a trade-off between good range/robustness and short ToA/high throughput [3].

 doi: 10.2313/NET-2024-04-1_13

5.3. Adaptive Data Rate (ADR)

One problem with LPWANs is the conflict between lower power consumption and wide range. A long transmission range also requires more energy in LoRaWAN, either caused by higher transmission power or a higher SF [7]. LoRaWAN's solution for this conflict is the ADR algorithm, which the NS performs. ADR tries for every ED to determine the proper communication parameters to enable reliable communication while prioritizing low energy consumption. This is done by dynamically adjusting the SF to the lowest SF possible while still maintaining a stable connection between the GW and the ED. A lower SF reduces the receiver sensitivity and the ToA, so transmissions have a lower power consumption and are less likely to collide [1]. ADR decision is based on the estimated link margin, calculated by measuring the SNR over the last few uplinks [16].

Each ED can decide on its own if it wants to use ADR or can only activate it if it detects transmission problems or deactivate it if the connection to the GW is stable. This enables LoRaWAN networks to adapt to changes in network infrastructure and to varying path loss, which allows EDs like battery-powered GPS trackers [4].

Current research has shown that LoRaWAN's implementation of ADR is not yet perfect. It was proposed to use a more sophisticated algorithm considering other objectives like scalability and throughput [17]. In [4], ADR was optimized to increase power efficiency by up to 25% and the packet success rate by nearly 7%. ADR can also cause problems even in small LoRaWAN networks where a few EDs have communication problems. Their SF and ToA will rise through the ADR algorithm, increasing communication problems and leading to network degradation, in which, ultimately, everyone uses SF12 [17]. The authors in [17] describe a different SF-management technique to avoid this problem, increasing the PDR up to 470%.

5.4. Energy Consumption

The energy consumption of LoRaWAN ED only depends on the ToA and transmission power, while the sleep current of the microcontroller of the ED can be neglected with μA at 3.3V in [7]. In [7], it was possible to only use 2.9mJ for a 23-byte transmission with SF7, bandwidth 125kHz, and a transmission power of 3dBm. For comparison, a standard 16850 Li-Ion battery with 3250 mAh has a capacity of about 43200000 mJ. To achieve the lowest power consumption, the following actions should be taken:

- Avoid using Semtech's PA Boost function, which increases the transmission power, for savings of up to 50% [7].
- Reduce the transmitter supply voltage. In [7], this was possible to lower the voltage from 3.3V to just 1.9V without reducing the transmission power, resulting in additional savings of 55%.
- Use newer generation transmitters [7].
- When higher transmission ranges are needed, the transmission power should be increased before raising the SF because of the significant negative impact of the SF on the ToA [7].
- Use smaller payload sizes and limit the amount of transmissions [1].

This shows that LoRaWAN is very power efficient and can be used for battery-powered devices with a lifetime of more than 10 years and is way more efficient than other standards like WIFI which uses about 90% more energy [1], [7], [18]. Another advantage of the low power consumption of LoRaWAN is that this allows to power the EDs using renewable energy sources, such as solar energy [18].

6. Scalability Problem

Scalability is a crucial aspect of IoT networks, and LoRa faces challenges due to its use of the ALOHA principle [1]. The ALOHA principle is a simple protocol where the EDs can transmit whenever they want, resulting in a high probability of collisions and retransmissions. This is especially a problem for LoRaWAN because of the long ToA and the duty cycle limits of the ISM band, resulting in a long block duration for the EDs and GWs [2].

Studies on the scalability of LoRa networks consistently suggest challenges in scaling, with notable sensitivity to increased network load [1]. This sensitivity manifests in a decrease in the packet delivery ratio (PDR) and an increase in network load due to retransmissions, primarily caused by collisions [2]. Simulations in [19] have shown that only 120 users per antenna can cause a PDR of only 90%. Classical collision avoidance mechanisms commonly used in wireless networks, such as Listen-Before-Talk using Channel Activity Detection (CAD) and closed-loop collision avoidance prove ineffective for LoRa due to specific characteristics, as discussed in [16].

6.1. Challenges in Existing Solutions

Attempts to address scalability challenges have encountered difficulties. Adaptive Data Rate (ADR) exhibits long convergence times, making handling increases in network density impractical. Enabling LoRaWAN's acknowledgment mode heavily increases network load and decreases the PDR of most nodes [2], [20]. Efforts to adapt well-known carrier sensing approaches for LoRa networks face reliability issues, particularly with Semtech's CAD, which becomes unreliable at distances less than 400 meters in dense urban environments. Slotted ALOHA or TDMA-like scheduling, while effective in low-density scenarios, struggle to scale due to high synchronization requirements and duty-cycle limitations [21].

6.2. Potential Solutions and Improvements

There are several strategies and improvements to mitigate the scalability challenges in LoRa networks:

- Directional antennas and multiple base stations can be advantageous in reducing communication interference [3].
- A new acknowledgment mode involving acknowledgment messages only for every N-received message and an instant message if a lost packet was

detected. This method was able to increase the PDR but not for all locations [20].

- Peer-to-peer mode, where nodes suffering from low PDR communicate with neighboring nodes for data forwarding, can overcome communication path issues. However, this may increase power consumption and only works if a neighbor in the range has a good connection to a GW [20].
- CANL LoRa, an open-loop collision avoidance mechanism employing a Listen-Before-Talk strategy, outperforms classical carrier sensing approaches in dense LoRa networks, as demonstrated in extensive simulations [16].

7. Conclusion

This paper provided insight into the functionality of LoRaWAN and its performance. LoRaWAN is a powerful LPWAN and stands out with its modulation technique and flexibility, which enables long-range communication with low power consumption, making it especially interesting for battery-powered EDs. LoRaWAN opens up many new possibilities for IoT devices, even if nearly only Class A devices are used today. Class B devices would make LoRaWAN even more attractive for actuators and partly solve LoRaWAN's scalability problem. ADR is a powerful tool to optimize the energy consumption of LoRaWAN EDs, but it is not perfect and can also not solve the scalability problem, which is the biggest problem of LoRaWAN and needs to be solved to enable the full potential of LoRaWAN.

References

[1] P. S. Cheong, J. Bergs, C. Hawinkel, and J. Famaey, "Comparison of LoRaWAN classes and their power consumption," in *2017 IEEE Symposium on Communications and Vehicular Technology (SCVT)*. IEEE, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8240313/

[2] H. E. Elbsir, M. Kassab, S. Bhiri, and M. H. Bedoui, "Evaluation of LoRaWAN Class B efficiency for downlink traffic," in *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, pp. 105–110. [Online]. Available: https://ieeexplore.ieee.org/document/9253405/

[3] Z. J. R. Kamoona and M. Ilyas, "Investigating the Performance of LoRa Communication for Nominal LoRa and Interleaved Chirp Spreading LoRa," in *2022 International Conference on Artificial Intelligence of Things (ICAIoT)*. IEEE, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/10121818/

[4] A. Ilarizky, A. Kurniawan, E. P. Subagyo, R. Harwahyu, and R. F. Sari, "Performance Analysis of Adaptive Data Rate Scheme at Network-Server Side in LoRaWAN Network," in *2021 2nd International Conference on ICT for Rural Development (IC-ICTRuDev)*. IEEE, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9656523/

[5] M. Abid Ali Khan, H. Ma, S. Muhammad Aamir, A. Baris Cekderi, M. Ahamed, and A. Abdo Ali Alsumeri, "Performance of Slotted ALOHA for LoRa-ESL Based on Adaptive Backoff and Intra Slicing," in *2022 6th International Conference on Communication and Information Systems (ICCIS)*. IEEE, pp. 169–173. [Online]. Available: https://ieeexplore.ieee.org/document/9998155/

[6] A. Griva, A. D. Boursianis, S. Wan, P. Sarigiannidis, G. Karagiannidis, and S. K. Goudos, "Performance Evaluation of LoRa Networks in an Open Field Cultivation Scenario," in *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9493416/

[7] E. Bäumker, A. Miguel Garcia, and P. Woias, "Minimizing power consumption of LoRa ® and LoRaWAN for low-power wireless sensor nodes," vol. 1407, no. 1, p. 012092. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1407/1/012092

[8] LoRa and LoRaWAN: Technical overview | DEVELOPER PORTAL. [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/

[9] C. Bouras, V. Kokkinos, and N. Papachristos, "Performance evaluation of LoraWan physical layer integration on IoT devices," in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8635715/

[10] Z. Xu, S. Tong, P. Xie, and J. Wang, "From demodulation to decoding: Toward complete lora phy understanding and implementation," *ACM Trans. Sen. Netw.*, vol. 18, no. 4, jan 2023. [Online]. Available: https://doi.org/10.1145/3546869

[11] L. Vangelista, "Frequency Shift Chirp Modulation: The LoRa Modulation," vol. 24, no. 12, pp. 1818–1821. [Online]. Available: http://ieeexplore.ieee.org/document/8067462/

[12] P. Edward, A. Muhammad, S. Elzeiny, M. Ashour, T. Elshabrawy, and J. Robert, "Enhancing the Capture Capabilities of LoRa Receivers," in *2019 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9069790/

[13] Y. Shiferaw, A. Arora, and F. Kuipers, "LoRaWAN Class B Multicast Scalability."

[14] European Telecommunications Standards Institute (ETSI), "ETSI EN 300 220-2 V3.2.1: Short Range Devices (SRD) operating in the frequency range 25 MHz to 1 000 MHz; Part 2: Harmonised Standard for access to radio spectrum for non specific radio equipment," ETSI, Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.02.01_60/en_30022002v030201p.pdf

[15] M. El-Aasser, T. Elshabrawy, and M. Ashour, "Joint spreading factor and coding rate assignment in lorawan networks," in *2018 IEEE Global Conference on Internet of Things (GCIoT)*, 2018, pp. 1–7.

[16] G. Gaillard and C. Pham, "CANL LoRa: Collision Avoidance by Neighbor Listening for Dense LoRa Networks," in *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, pp. 1293–1298. [Online]. Available: https://ieeexplore.ieee.org/document/10218282/

[17] L. Casals, C. Gomez, and R. Vidal, "The SF12 Well in LoRaWAN: Problem and End-Device-Based Solutions," vol. 21, no. 19, p. 6478. [Online]. Available: https://www.mdpi.com/1424-8220/21/19/6478

[18] A. Kurtoglu, J. Carletta, and K.-S. Lee, "Energy consumption in long-range linear wireless sensor networks using LoRaWan and ZigBee," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, pp. 1163–1167. [Online]. Available: http://ieeexplore.ieee.org/document/8053135/

[19] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso, "Do lora low-power wide-area networks scale?" in *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 59–67. [Online]. Available: https://doi.org/10.1145/2988287.2989163

[20] S.-Y. Wang and T.-Y. Chen, "Increasing LoRaWAN Application-Layer Message Delivery Success Rates," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, pp. 00 148–00 153. [Online]. Available: https://ieeexplore.ieee.org/document/8538457/

[21] N. Kouvelas, R. V. Prasad, N. Yazdani, and D. E. Lucani, "np-cecada: Enhancing ubiquitous connectivity of lora networks," in *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, 2021, pp. 374–382.

Covert Communication over ICMP

Georgios Merezas, Lars Wüstrich*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: georgios.merezas@tum.de, wuestrich@net.in.tum.de

***Abstract*—In the ever-expanding landscape of the internet, robust security measures are critical components of any modern technology. As our dependence on digital networks continues to grow, so does the sophistication of malicious actors seeking to exploit vulnerabilities that arise in all complex systems. One of the many tools of these malicious actors is ICMP tunneling, a method used for the establishment of hidden channels within network environments. These channels allow hidden and unfiltered communication of infected machines with potential attackers. The paper explores the structure of ICMP packets and how it facilitates them to tunnel TCP connections. Furthermore, it examines some existing applications of ICMP tunneling, like botnets. Finally, it proposes some countermeasures to ICMP tunneling.**

***Index Terms*—ICMP tunneling, covert channels, botnets**

1. Introduction

In the increasingly interconnected world that we live in, the internet has become an integral part of our lives. As everything in our lives becomes digitalized, the interest of malicious actors in stealing and exploiting digital data also increases. Command and Control (CnC) techniques are used to send commands and receive data from infected machines, which allows an attacker to continuously communicate with them [1]. This can be lucrative when one can steal business secrets or mine digital currency on compromised devices [2]. State actors, too, have a use for remote access to machines, either to spy or to launch Distributed Denial of Service (DDoS) attacks against foreign businesses as a form of economic protectionism [3].

One way to achieve remote access to a host and keep the communication hidden is to use ICMP tunneling [4]. This paper should provide insight into cybersecurity challenges that were first reported in 1997 [4] [5]. Additionally, it should highlight how simple network status messages can be exploited in unintended ways. The contribution of this paper is: (a) an explanation of ICMP tunneling, (b) an exploration of its applications, (c) a summary of existing proposals to prevent ICMP tunneling.

Section 2, introduces ICMP and ICMPv6, with their standard use cases. In Section 3, we first explain how ICMP tunneling works and then outline its capabilities. In Section 4, existing applications of ICMP tunneling are explored. Section 5, introduces preventative measures to combat ICMP tunneling.

2. Background

Internet Protocol (IP) is a network layer protocol, that forms the basis of the Internet. It is used to transmit IP packets globally between hosts. An IP packet consists of an IP header and IP data. IP data, i.e. the content that is transmitted by a given packet, is appended after the header (encapsulated within the IP packet). Such data is for example ICMP or higher layer protocols. The IP header specifies the source and destination addresses, if it is a fragment, packet length, and other packet information. [6]

User Datagram Protocol (UDP) is a simple transport layer protocol. It specifies the source and destination ports of the datagram and a checksum for error detection. The ports of UDP can be used to multiplex the communication between two IP hosts. The only feature of UDP that can be used for data loss, is the fact that error messages about IP packets are sent when a packet is lost in transit. [7]

Transmission Control Protocol (TCP) is a widely implemented transport layer protocol. Similar to UDP it has ports for multiplexing and a checksum for error detection. However, TCP has a distinct advantage over UDP in that it builds the connection at the start of the communication and destroys it at the end. Most importantly, during the communication, it acknowledges segments it has received (a segment being sent as an IP packet). This feature builds the basis for congestion and flow control. When the sender does not have enough acknowledgments for the segments that have already been sent, it starts sending them at a slower rate to avoid possible congestion in the network between it and the receiver. When the sender receives an acknowledgment for a fragment that was sent too long ago or does not receive an acknowledgment in some predetermined time, it tries to resend it. This means that a TCP connection is reliable, and can fail only when a packet can never reach its destination, e.g. in the event of a complete network shutdown. [8]

Internet Control Message Protocol (ICMP) is a network layer protocol part of the IP suite. It is primarily used for error reporting and the exchange of control information. ICMP messages are encapsulated within IP packets as if they were a higher-level transport protocol, like TCP or UDP. However, they are an integral part of IP and are processed as a special case. Unlike TCP and UDP, ICMP packets do not specify ports. [9]
ICMP is used by network utilities like `ping` [10] to send ICMP Echo Requests and receive corresponding ICMP Echo Reply messages. It is used to test if a host is reachable on an IP network. Another utility, `traceroute` [11], sends out many IP packets after each other, incrementing

 doi: 10.2313/NET-2024-04-1_14

their Time-To-Live (TTL). The TTL is decremented by each router on the path to the destination. When the TTL of a packet reaches 0, the last reached router sends an ICMP Time Exceeded message to the sender. This way, the utility maps the path between the source machine and a remote host.
An ICMP messages consist of a header and content sections. The header is 4 bytes (B) long. It contains the type (1B), the code (1B), and the checksum (2B). The type specifies if the ICMP message is an ICMP Echo Request, an ICMP Echo Reply, an ICMP Time Exceeded message, etc. The code provides further information about the type, e.g., why time was exceeded. The checksum is used for error checking. The content section varies depending on the type and code of the ICMP message, and it has no preset length.
ICMPv6 is the version of ICMP used in conjunction with IPv6. It serves similar purposes to ICMP in IPv4, except for some enhancements to adapt to the features of IPv6. For example, ICMPv6 has Neighbor Discovery Protocol (NDP) instead of the ARP protocol in IPv4. ARP is used to find the Layer 2 address of devices whose Layer 3 (IP) address is known. NDP has the same functionality, in addition to many other improvements. Another example is an ICMPv6 Packet Too Big message. It exists because only the sender of the IPv6 packet can fragment it, whereas IPv4 packets can be fragmented by any router on the way to the destination.
Because the principal idea and structure of ICMPv6 is the same as ICMPv4, in this paper the explained principles of ICMP tunneling will be relevant for both IPv4 and IPv6 communication.

3. ICMP Tunneling

ICMP tunneling can be accomplished with any type of ICMP message [4]. Most tools use ICMP Echo Request or ICMP Echo Reply packets, which are normally used to test host reachability. This makes it hard to filter against malicious uses and, therefore, very appropriate to use for hidden communication. [12]

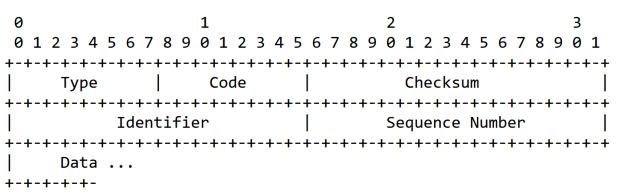


Figure 1: ICMP Echo Request and Reply as defined in RFC792 [9]

3.1. Prerequisites

The structure of ICMP allows for varied length data after the header for type 0 (Echo Reply) and type 8 (Echo Request) ICMP packets, as can be seen in Figure 1. This is defined in RFC792 [9] to allow for flexible network testing. For example, big ICMP Echo Request packets can be sent to test network load [4]. There are no specifications for the type of content in the data section of ICMP Echo Request/Reply packets. This enables malicious actors to use ICMP to transmit data to infected machines with a smaller network footprint that may go unnoticed by firewalls and other detection tools [12].

For ICMP tunneling to work, root access is needed [13]. For more rudimentary programs than the ones discussed further, root access is not always necessary. The main prerequisite is that the tunneling program is installed on the device that will be used for communication. This can be either because it was infected by a malicious actor, or due to the user of the machine wishing to hide their communication from the local system administrator.

3.2. How it works

Using ICMP Echo Request and Reply packets, a malicious attacker can embed information into the data section and achieve communication. An infected machine can listen for incoming requests and read the data that was sent. It can then confirm with an ICMP Echo Reply that the message was received.

3.2.1. Functional principle. ICMP tunneling does not work out of the box simply by using kernel sockets. A separate program needs to be written to facilitate this communication. Such a program should be able to embed the required content into ICMP packets and send them. It then should be able to listen for incoming ICMP packets and read the content of those participating in the disguised communication. It should also be able to process the embedded data and continue communicating.

An example of a program that communicates using ICMP tunneling can embed TCP/UDP content into the data section of ICMP Echo Request/Reply packets. As most programs are already created to work with either TCP or UDP, the two main transport layer protocols, such an implementation can have wide applications.

It can be designed as a tool that intercepts some program's outgoing packets or all packets on some port. It then translates them into ICMP Echo Request packets by embedding each transport layer part of the packet into the data section, before sending it out. The same tool on the infected machine listens for ICMP Echo Requests with certain predefined features that mark it as a tunneled connection (e.g. a specific predefined Identifier). Then, it intercepts the ICMP packet, extracts the TCP/UDP content, and sends that to a process on the same machine. Thus, a TCP/UDP packet is transmitted between two machines with ICMP. Essentially, such a tool has the implementation of a kernel socket with the addition of embedding the packet into an ICMP Echo Request. Ptunnel [13] is a program that uses this concept. [12]

This type of ICMP tunneling requires administrative or root privileges to be able to run. The aforementioned ICMP tunneling tool Ptunnel, for example, requires root access to be able to use raw sockets, i.e., without specifying if they are TCP or UDP [13]. This makes it harder to achieve the end goal, but not overall impossible.

3.2.2. Multiplexing. Figure 1 shows that ICMP Echo Request/Reply packets contain an Identifier and a Sequence Number. These can mark a packet as a tunneled connection, as mentioned previously. Additionally, they

 doi: 10.2313/NET-2024-04-1_14

can be used to multiplex the communication, i.e., have parallel streams that contain data for different purposes. The hosts that are performing the communication can match an incoming ICMP Echo Reply to an ICMP Echo Request they made and, therefore, know which stream this packet belongs to. Usually, ICMP Echo Request/Reply communication already happens this way in, for example, the `ping` utility.

Additionally, embedding UDP/TCP content in the data section can lead to more parallel streams. For example, one could treat the Identifier concatenated with the source port of UDP as the 'source' and the Sequence Number concatenated with the destination port of UDP as the 'destination'. The ports of UDP and TCP are 2B or 16 bits long [8] [7], and so are the Identifier and Sequence Number (see Fig. 1). In such a scheme, the port numbers could go up to $2^{(16+16)}$ or 2^{32}, instead of just 2^{16} as they are in TCP and UDP. Such high numbers would allow for greater multiplexing capabilities.

3.3. Capabilities

While ICMP packets are similar to UDP and TCP, they have distinct features that impact the quality of the communication between hosts. In this analysis, we assume that the ICMP communication is performed using ICMP Echo Request and ICMP Echo Reply packets that embed UDP or TCP into the data section, which can be seen in Figure 1.

3.3.1. Payload size. The size of an ICMP Echo Request or Reply packet, outside artificial size restrictions that may be imposed by the OS or a firewall, is only limited by the size of the IP packet that it is part of. For Ethernet communication, the Maximum Transmission Unit (MTU) size is typically 1500B [14]. The MTU defines the largest possible size for the contents of the Ethernet frame, i.e., the maximum size of a single IP packet. The IP packet itself has a minimum header size of 20B [6]. A TCP header has a minimum size of 20B [8], while UDP is only 8B [7]. That means the maximum content size for TCP is 1460B and for UDP 1472B. With the addition of the ICMP header between the IP and TCP/UDP headers, the maximum content size is reduced by 4B. Additionally, due to the use of ICMP Echo Request/Reply packets, we need to account for the 4B of the Identifier and the Sequence Number. So, the final content that we can transmit has a size of 1452B for a tunneled TCP packet and 1464B for a tunneled UDP packet. This is only 8B less than the standard TCP/UDP communication.

3.3.2. Reliability. ICMP packets do not differ significantly from UDP packets. They both lack flow or congestion control, unlike TCP. The ports of UDP can be simulated with the Identifier and Sequence Number of ICMP Echo Request/Reply packets. However, they have one major difference. ICMP error messages are not sent about other ICMP messages to prevent "the infinite regress of messages about messages, etc." [9]. This has consequences for the reliability of ICMP communication. If an ICMP message is lost, the sender will never be notified about it and, therefore, cannot retry.

In the given schema, we do not have normal ICMP Echo Request/Reply packets. By embedding TCP or UDP content into the data section, we can use features of these transport layer protocols that can affect the reliability of the communication. If we use UDP, we do not achieve more reliability because UDP does not have mechanisms for tracking sent and received segments. If we embed TCP, we can use its features to create a unique, reliable ICMP communication protocol.

The reliability features of TCP explained in Section 2 can be implemented by an ICMP tunneling tool to make the connection reliable. This transmission channel has the disadvantage of not receiving error messages when packets inevitably get lost in transit. Otherwise, it acts just like a standard TCP connection. That is because usual IP communication does not guarantee that a control message is returned in case an IP packet is not delivered [9]. Some failure needs to happen twice for the sender to not retry to send a packet after it fails: when sending the IP packet and when the ICMP error packet is sent back. In our case, only one failure needs to occur for a packet to be lost. This means that packets are resent more often, and the average segment rate is lower.

4. Applications

The use of ICMP tunneling for communication has the advantage of being hard to detect and filter against by firewalls and Intrusion Detection Systems (IDS) [12]. Logically, most applications of ICMP tunneling are those that should not be detected by third parties or are performed by malicious actors. These are so-called covert channels.

A covert channel is a communication channel that is "not intended for information transfer at all" [15]. Therefore, applications that can or do use ICMP tunneling use it as a covert channel.

A prime example of a covert channel is communication with a backdoor on an infected system. This can provide complete access to the system without the firewall blocking the channel. A more subtle covert channel would be having an infected computer with software that sends out stolen confidential data at random intervals [4].

4.1. Botnets

One of the ways that malicious actors use covert channels is for CnC communication of botnets [1]. Botnets are networks of infected computers that each run one or more bots. They are created and controlled by a botmaster who issues commands to these bots. These bots were usually centrally controlled using the IRC messaging protocol [16]; nowadays, botnets are more often deployed as peer-to-peer networks due to more sophisticated detection methods and to avoid having one point of failure. These collections of infected computers can communicate with each other, perform DDoS attacks, and send spam. They can also allow the attacker to have remote access to all of the affected devices. Some examples of BotNets include Mirai [17] [18], Mariposa [19], and others. [20] [21]

A botnet that uses ICMP tunneling is Pingback. Unlike the discussed implementation, the malware does not embed TCP into ICMP for communicating, rather it has

 doi: 10.2313/NET-2024-04-1_14

its own data in ICMP Echo Request packets, as can be seen in Figure 2. The malware is hidden inside a malicious `oci.dll` file. This file is normally loaded with two other `.dll` files by the `msdtc` Windows service. This service loads an ODBC library to support Oracle databases. The library tries to load three Oracle ODBC DLLs, one of which is the `oci.dll`. When the malware is running, it listens for ICMP Echo Request packets with sequence numbers 1234, 1235 or 1236, and can execute shell commands remotely. [22]

```
struct ICMPData {
   char  cmd[10];          // bot command (see appendix for more details)
   char  args[512];        // extra parameter, but haven't seen it used by the malware
   char  cmd_line[258];       // command line(see appendix for more details)
   unsigned long  dest_port;  // destination port
   char  dest_addr[4];        // destination IP address
}
```

Figure 2: Pingback data struct, published in [22]

4.2. Data theft

Another use of covert channels is the breach of data confidentiality. An attacker can extract data from a machine and send it out over a covert channel by infecting it. Thus, private and confidential data is stolen without any detection by the user or the network administrator. This application is particularly harmful when it comes to state actors performing espionage on foreign governmental organizations [3]. Similarly, businesses can commit illegal economic espionage and acquire business secrets to gain a competitive market advantage. Criminal organizations can steal private information and blackmail its owners into paying them ransom.

A practical application is Cobalt Strike [23], a commercial remote access tool that is used to "execute targeted attacks and emulate the post-exploitation actions of advanced threat actors." Amongst many features, it has the capability to communicate with ICMP.

4.3. Integrated systems

Covert channels can also infiltrate integrated systems that were previously offline [24]. For example, systems in self-driving cars, or Internet of Things (IoT) devices, like home assistants or security cameras. These networks are based on the IP suite and are, therefore, vulnerable to ICMP tunneling, amongst other covert channels. Their cybersecurity is usually much more lax than that of more sophisticated systems. This is due to their widespread use allowing for greater possibilities of social engineering, and the hardware vulnerabilities arising from the lesser focus on security during their development [25]. Attacks against them include the threat of spying through cameras and microphones. More dangerous threats have the horrible potential of ending multiple peoples' lives if the attacker can gain unrestricted access to the mechanical controls of a car.

5. Prevention

There are preventative measures that limit the possibilities for communication using an ICMP tunnel [4]. Each one of them can be circumvented, or sometimes, a measure can disrupt the normal user [4]. A sensible combination of them can prevent all but the most sophisticated attacks.

The main prerequisite for ICMP tunneling is the variable data of ICMP Echo Request/Reply packets, both in content and in length. Operating systems have preset lengths and content for the ICMP Echo Request packets that they generate. For example, Linux has a standard data section length of 56B for ICMP Echo Requests, while Windows has 32B [5]. An IDS can filter against all ICMP packets that do not match the content of these two standards. Thus, the attacker needs to fragment the data into more ICMP packets to pass the filter. Sometimes, though, such a filter is disruptive. Big ICMP Echo Request packets are helpful to test whether a network is capable of carrying them [4]. Inspecting large packets for suspicious content is even more difficult, especially if the covert channel uses encryption. Determining if something is encrypted is not always a fail-safe method [4].

Another way to limit the communication over ICMP packets is to have stateful firewalls or NAT devices [4]. These track all ICMP traffic. If an ICMP Echo Request is sent, the identifier and sequence number are saved. Only a matching ICMP Echo Reply is let through to the original host. Other firewalls create their own ICMP packets that mirror the ones sent out by the host but with their own data. If they receive an ICMP Echo Reply, they then create an ICMP Echo Reply that matches the ICMP Echo Request sent by the host. This effectively disrupts the communication channel.

There also exists a proposal for a kernel module that scans ICMP packets for any malicious content [4]. This solution was proposed with the assumption that stateful firewalls are too resource-heavy to be implemented on personal machines. It was tested on 2003 hardware and had acceptable performance at the time. Nowadays, it should have very minimal overhead.

Another complex proposal is presented by Sayadi [5]. The detection includes two stages. In stage 1, the following three steps are performed on an ICMP packet: (a) the packet is checked for preset Linux and Windows lengths, (b) the method tries to match the packet against only one existing ICMP Echo Requests, (c) it is checked if there is an absence of a spike of ICMP messages. If all checks, performed after each other, succeed, then the message is regarded as normal. If either of the three points fails, then Stage 2 is triggered. Stage 2 tries to randomly pattern match against the standard Linux and Windows content. If it fails, the message is regarded as part of a covert channel.

The evolving field of machine learning can also be used to filter out covert channels in ICMP [26]. The proposal by Cho [27] is a promising new addition to existing tunneling prevention methods that use machine learning. Their algorithm has a 99.9% accuracy in detecting covert channels, an improvement over older proposals [28], [29].

All of the above methods have different approaches to dealing with a tunneled ICMP connection. The proposals on machine learning are promising, and their universality needs to be explored further [26]. A balanced approach of defensive methods needs to be taken, as at-scale implementation of machine learning results in many false positives [26].

6. Conclusion

ICMP tunneling is used for establishing covert channels on IP networks. Using fact that data of the ICMP Echo Request/Reply packets is not standardised in general, a tool can embed its own content into ICMP packets. An example application can embed TCP content, thus trasmiting TCP packets without the network administrators observing it. This connection can be multiplexed using fields of the ICMP Echo Request/Reply header and the TCP ports. The reliability of such a communication is comparable to normal TCP, with the packet rate being slightly slower and dependent on the rate of packet loss.

Since ICMP tunneling is used to hide communication on networks, a tunneled connection between two hosts results in a covert channel. Applications that make use of covert channels can use ICMP tunneling to achieve it. Some existing examples include botnets, like Pingback, and remote access backdoors, like Cobalt Strike.

There are proposed theoretical solutions to prevent ICMP tunneling, but practical applications are lacking. Many proposals exist that have been tested in simulated environments and perform very well. The field of machine learning also has a lot of promising research, which however still needs to be practically implemented on a wide scale. Further research of practical applications of these preventative measures and how they can be effectively combined together has to be conducted, to ensure that ICMP tunneling is not a threat to modern networks.

References

[1] D. D. Jovanović and P. V. Vuletić, "Analysis and Characterization of IoT Malware Command and Control Communication," in *2019 27th Telecommunications Forum (TELFOR)*, 2019, pp. 1–4.

[2] H. Dhayal and J. Kumar, "Botnet and P2P Botnet Detection Strategies: A Review," in *2018 International Conference on Communication and Signal Processing (ICCSP)*, 2018, pp. 1077–1082.

[3] J. Ford and H. S. Berry, "Leveling Up Survey of How Nation States Leverage Cyber Operations to Even the Playing Field," in *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*, 2023, pp. 1–5.

[4] A. Singh, O. Nordström, C. Lu, and A. L. M. dos Santos, "Malicious ICMP Tunneling: Defense against the Vulnerability," in *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, 2003, pp. 226–236.

[5] S. Sayadi, T. Abbes, and A. Bouhoula, "Detection of Covert Channels Over ICMP Protocol," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, 2017, pp. 1247–1252.

[6] J. Postel, "Internet Protocol," RFC 791, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/info/rfc791

[7] J. Postel, "User Datagram Protocol," RFC 768, Aug. 1980. [Online]. Available: https://www.rfc-editor.org/info/rfc768

[8] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[9] J. Postel, "Internet Control Message Protocol," RFC 792, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/info/rfc792

[10] "ping(8) - Linux man page," https://linux.die.net/man/8/ping, [Online; accessed 2024-03-02].

[11] "traceroute(8) - Linux man page," https://linux.die.net/man/8/traceroute, [Online; accessed 2024-03-02].

[12] K. Stokes, B. Yuan, D. Johnson, and P. Lutz, "ICMP Covert Channel Resiliency," in *Technological Developments in Networking, Education and Automation*, K. Elleithy, T. Sobh, M. Iskander, V. Kapila, M. A. Karim, and A. Mahmood, Eds. Dordrecht: Springer Netherlands, 2010, pp. 503–506.

[13] D. Stødle, "Ping Tunnel," https://www.cs.uit.no/~daniels/PingTunnel/, 2004, [Online; accessed 2023-12-17].

[14] "IEEE Standard for Ethernet," *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pp. 1–7025, 2022.

[15] B. W. Lampson, "A Note on the Confinement Problem," *Commun. ACM*, vol. 16, no. 10, p. 613–615, oct 1973. [Online]. Available: https://doi-org.eaccess.tum.edu/10.1145/362375.362389

[16] J. Govil and J. Govil, "Criminology of BotNets and their detection and defense methods," in *2007 IEEE International Conference on Electro/Information Technology*, 2007, pp. 215–220.

[17] "Mirai BotNet Source Code," https://github.com/jgamblin/Mirai-Source-Code, 2016, [Online; accessed 2024-02-27].

[18] G. Gallopeni, B. Rodrigues, M. Franco, and B. Stiller, "A Practical Analysis on Mirai Botnet Traffic," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 667–668.

[19] P. Sinha, A. Boukhtouta, V. H. Belarde, and M. Debbabi, "Insights from the analysis of the Mariposa botnet," in *2010 Fifth International Conference on Risks and Security of Internet and Systems (CRiSIS)*, 2010, pp. 1–9.

[20] M. Singh, M. Singh, and S. Kaur, "TI-16 DNS Labeled Dataset for Detecting Botnets," *IEEE Access*, vol. 11, pp. 62 616–62 629, 2023.

[21] "List of Botnets," https://netacea.com/glossary/list-of-botnets/, 2021, [Online; accessed 2024-02-27].

[22] L. Macrohon and R. Mendrez, "Pingback: Backdoor At The End Of The ICMP Tunnel," https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/backdoor-at-the-end-of-the-icmp-tunnel/, 2021, [Online; accessed 2024-02-27].

[23] "Cobalt Strike," https://attack.mitre.org/versions/v11/software/S0154/, 2017, [Online; accessed 2024-02-27].

[24] A. Ondov and P. Helebrandt, "Covert Channel Detection Methods," in *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2022, pp. 491–496.

[25] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-Defined Security," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 250–10 276, 2020.

[26] Z. Sui, H. Shu, F. Kang, Y. Huang, and G. Huo, "A Comprehensive Review of Tunnel Detection on Multilayer Protocols: From Traditional to Machine Learning Approaches," *Applied Sciences*, vol. 13, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/3/1974

[27] D. Cho, D. Thuong, and N. Dung, "A Method of Detecting Storage Based Network Steganography Using Machine Learning," *Procedia Computer Science*, vol. 154, pp. 543–548, 2019, proceedings of the 9th International Conference of Information and Communication Technology [ICICT-2019] Nanning, Guangxi, China January 11-13, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919308555

[28] A. N. Mahajan and I. Shaikh, "Detect covert channels in TCP/IP header using Naive Bayes," *International Journal of Computer Science and Mobile Computing*, vol. 4, pp. 881–883, 2015.

[29] T. Sohn, J. Seo, and J. Moon, "A study on the covert channel detection of TCP/IP header using support vector machine," in *Information and Communications Security: 5th International Conference, ICICS 2003, Huhehaote, China, October 10-13, 2003. Proceedings 5*. Springer, 2003, pp. 313–324.

Literature Survey: Performance Enhancing Proxies for TCP and QUIC

Ahmed Rayen Mhadhbi, Lion Steger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: ge95saq@tum.de, stegerl@net.in.tum.de

***Abstract*—Transmission Control Protocol (TCP), the dominating transport protocol in the Internet, was originally mainly designed for wired networks where packet loss is mostly due to congestion. Deploying standard TCP in hybrid networks leads to a performance degradation due to the wireless link characteristics. To deal with this issue, intermediary nodes on the network path called Performance Enhancing Proxies (PEPs) were designed. In this paper, we first survey PEPs for TCP to assess their relevance in the current research. We then focus on current PEP approaches for QUIC, a new transport protocol incompatible with TCP PEPs. We find practical approaches that benefit the performance of QUIC and discuss their limitations mostly regarding scalability and security.**

***Index Terms*—performance enhancing proxies, satellite networks, tcp, quic, masque**

1. Introduction

TCP is a transport layer protocol that provides a reliable connection-oriented ordered data transfer while offering mechanisms for flow and congestion control [1]. It was originally designed mainly for wired networks where packet losses are mostly due to network congestion. Standard TCP interprets packet losses always as congestive and reduces the data transfer rate unnecessarily in case of high *Bit Error Rate* (BER) which leads to a significant performance degradation.

This unwanted behavior is observed in networks that contain both wired and wireless connections, called *hybrid networks*. The wireless section is in particular characterized by a high BER. TCP has to adapt to the different needs of these networks to provide the required performance. To this end, several enhancement approaches have been proposed: Wireless TCP variants that introduce modifications to the standard protocol were designed and tailored for the specific needs of the different wireless networks [2]. Other approaches introduced intermediary middleboxes on the network path called *Performance Enhancing Proxies* (PEPs) [3]. The common approach is to split the TCP connection transparently into two independent connections. Congestion and error control can therefore be tailored for the specific characteristics of each connection.

The development of QUIC [4] as a secure transport protocol based on end-to-end encryption of not only the payload but also most of the control information renders transparent PEPs useless, since they rely on the inspection of unencrypted TCP headers for their functionality. However, the benefits of connection splitting for performance enhancement motivate the exploration of PEPs for QUIC [5].

In this paper, we first provide a general overview of the challenges posed by hybrid networks, an explanation of PEPs' functionality and QUIC. We then investigate TCP PEPs. Finally, we present, analyze and discuss the limitations of different approaches to QUIC PEPs gleaned from the current research.

2. Background

We first provide a general overview of the challenges posed by hybrid networks, an explanation of PEPs' functionality and QUIC.

2.1. Problems of Wireless Networks

There are various wireless network architectures. We mention for instance *satellite networks* and *cellular networks*. Despite the challenges posed by their specific needs, all wireless networks face the problem of high BER since thy use air as transmission medium and are therefore more prone to random factors, such as bad weather conditions and the mobility of end users [2]. In cellular networks, a base station interconnects the wired fast network and the wireless mobile network to provide internet access for mobile users. Such infrastructure is characterized by frequent handoffs and low wireless bandwidth. Satellite networks are of paramount importance in today's internet communication. They provide connectivity for ships [6], planes, users in less populated areas and in times of disaster. Traditional geostationary (GEO) satellites are widely used despite the rising of new technologies such as Low Earth Orbit (LEO) satellites. Staying at an altitude of 35'786km, GEO satellite communication results in a high round trip time (RTT) of about 600 ms leading, coupled with local packet losses, to drastic goodput degradation. PEPs, which we introduce in the following are, among other approaches [7], deployed to tackle this problem.

2.2. Performance Enhancing Proxies

PEPs are intermediary nodes on the communication path designed to improve the performance of TCP [3]. They can operate at link layer, transport layer and application layer. At transport layer, PEPs split the TCP connection in two separate connections: The first one is terminated at the PEP and the new one is established

 doi: 10.2313/NET-2024-04-1_15

from the PEP to the server or another PEP. They can be deployed as integrated PEPs: A single PEP splitting the connection or distributed PEPs: Two PEPs isolate the link between them. Connection splitting leads to a shorter RTT for both connections which increases the TCP sender transmission rate as the time to get *acknowledgements* gets shorter. It also enables the PEP to apply congestion and error control tailored for the specific needs of the network segment providing significant performance gains in wireless networks [3]. Despite their wide deployment [8] and advantages, connection-splitting PEPs violate the end-to-end semantics of TCP and contribute to the *ossification of the transport layer* [9] [10]: Their functionality is based on assumptions about TCP headers and the protocol operations so a new update to the protocol becomes difficult since it requires a modification of their design when they are already deployed in the Internet.

2.3. QUIC

QUIC is a transport protocol originally developed by Google [4] and standardized by the IETF [11]. It is based on the end-to-end encryption of not only the payload but also most of the control information [12]. Some of its main advantages are facilitating the process of rolling out updates due to its user space implementation, stream multiplexing to support multiple streams within a single connection avoiding the head-of-line blocking of TCP, reducing handshake overhead by making use of 0-RTT and circumventing the ossification of the protocol, being based on UDP traffic [4].

3. TCP PEPs

In this section, we will investigate PEPs for TCP. We first succinctly present a number of approaches we do not consider to be recent. We then focus on more recent ideas.

3.1. Overview of PEP Implementations

The benefits of PEPs performing connection splitting have been investigated in multiple environments. XCP-PEP [13] is a transparent PEP that leverages the eXplicit Control Protocol (XCP), developed as a new congestion control mechanism [14] to enhance performance over a satellite link. Experiments showed a fast end-to-end link utilization and a quick fairness convergence when deploying XCP-PEP in high latency networks such as satellite networks. HTTPPEP [15] is another PEP example that provides a faster web browsing experience when using a satellite based network by applying protocol and transport layer optimizations coupled with data compression. In mobile systems such as Long Term Evolution (LTE) networks, performance gains in web browsing and video streaming using PEPs have been observed [16]. We focus on some of the ideas in the following.

3.2. TCP Performance Enhancement over Satellite Networks

Satellite networks pose several challenges to TCP given their characteristics. PEPs were employed among other approaches to improve TCP performance over satellite networks. PEPsal was developed by Caini et al. [17] as the first open source integrated splitting approach available for Linux OS under the GNU GPL license. It is a TCP PEP approach that improves the performance of the satellite connection by employing TCP Hybla, a TCP enhancement scheme specifically designed for the needs of satellite networks [18]. Evaluating PEPsal in the presence of congestion and link losses showed performance gains. Some of the results can be attributed to using TCP Hybla on the satellite segment [17]. Although PEPsal is not a recent TCP PEP, we highlight its relevance and wide usage in research.

3.3. Translation between Network Architectures

The deployment of new network architectures to solve the issues faced by the current TCP/IP architecture due to the exponential growth of data traffic is met by the inflexibility of the internet infrastructure [19]. To circumvent this issue, the idea of *translating* between the existing architecture and a new one was thoroughly investigated by Ciko et al. [20], introducing a Performance Enhancing Proxy for Deploying Network Architectures (PEP-DNA).

PEP-DNA is the first TCP PEP developed with the goal of enabling sending data from TCP/IP applications along a network path with a different underlying architecture and in the other direction translating traffic to be compatible with the TCP/IP architecture. It is fully implemented in kernel space to be deployed on Linux. It has been tested for the translation in a TCP-TCP connection, between TCP and Recursive InterNetwork Architecture (RINA) [21] as well as between TCP and an Information Centric Networking (ICN) architecture [22].

The performance evaluation of PEP-DNA showed its efficiency and scalability achieving good throughput with low CPU and memory utilization. These results make the deployment of new network architectures a realistic objective. Future work should investigate the possibility of PEP-DNA supporting other protocols besides TCP, rendering the proxying explicit to allow a consent based translation and using dynamic proxying mechanisms [20].

3.4. Multi-Domain Congestion Control

Applying different congestion control mechanisms optimized for the different domains in hybrid networks leads to a performance improvement. The concept of Multi-Domain Congestion Control (MDCC) [23] could be achieved through the deployment of transparent connection splitting PEPs that apply an appropriate *Congestion Control* (CC) for each connection. However, traditional PEPs introduce a processing overhead and contribute to the ossification of the transport layer. Approaches applying end-to-end encryption render transparent PEPs impractical. In order to deploy PEPs without violating the end-to-end semantics, Middlebox Cooperation Protocols (MCPs) [24] propose the idea of middleboxes sharing explicit useful information to the endpoints that can be safely ignored. Based on this concept, Mihály et al. [25] developed a lightweight PEP (LwPEP) that supports MDCC, overcoming the additional communication and processing

overhead and circumventing the problem of transport ossification without modification at the client side. LwPEP enabled the design of a MDCC showing performance gains in LTE [25] and mmWave 5G Networks [26]. LwPEP supports both TCP and QUIC traffic [26].

4. QUIC PEPs

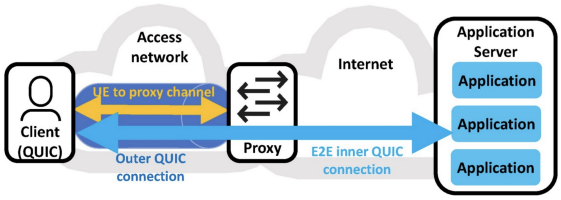


Figure 1: MASQUE proxy setup with QUIC tunneling [27])

We first start by addressing the fundamental question of whether exploring the possibilities for QUIC PEPs is justifiable in terms of potential performance gains. Kosek et al. [5] investigated this question for satellite networks using both GEO and LEO satellites, comparing QUIC vs. TCP and HTTP/3 vs HTTP/1.1 with and without PEPs.

Evaluating the performance regarding goodput over time showed a better performance when using PEPs for both GEO and LEO scenarios with different link loss rates, especially with higher RTTs and lower loss rates. Analysing the web performance also demonstrated a real performance gain using QUIC PEPs in GEO orbits. These promising results motivate the further exploration of QUIC PEPs. However, the used implementation in [5] is a proof-of-concept: The developed proxy for connection splitting operates on clear data, thereby violating the principle of end-to-end encryption. We therefore present the following practical possibilities for QUIC PEPs with the common goal of preserving end-to-end encryption. For each discussed concept, we highlight the important points, give an overview of the implementation and discuss the limitations.

4.1. Explicit User-consent based Proxying

Motivation: As transparent PEPs are incompatible with QUIC, explicit proxying should be investigated. In this context, we introduce Multiplexed Application Substrate over QUIC Encryption (MASQUE) [28]. It is a QUIC proxying protocol that defines a new HTTP CONNECT extension to enable a tunneled QUIC connection between a client and a proxy, as illustrated by Figure 1: An end-to-end connection over the proxy encapsulated in an outer QUIC tunnel connection. By using MASQUE, the client can explicitly request forwarding of UDP and IP traffic towards a specific server. The proxy receives packets from the client wrapped in an outer QUIC tunnel connection, unwraps them and sends them to the target server. The proxy operates in the other direction as well by encapsulating received packets from the server and relaying them to the client. MASQUE replaces transparent PEPs that operate transparently with explicit user-consent based proxying [28] and provides web proxying without interception and end-to-end security problems which some HTTP based explicit proxies deployed in access networks might suffer from [29].

Implementation: MASQUE is still under development. The IETF MASQUE working group developed two specifications for defining a new method to proxy UDP in HTTP. The first is using CONNECT UP HTTP method [30] to create a tunnel to a proxy server over the HTTP request stream to enable sending tunneled UDP payloads using HTTP datagrams to the proxy. The second specification [31] describes two ways of data transmission: using QUIC datagrams [32] for unreliable data transfer called *datagram mode* or using datagrams encoded as CAPSULE frames [30], a new HTTP type frame, for reliable transmission called *stream mode*.

Evaluation: The impact of using QUIC based MASQUE proxying on QUIC performance has been studied by Kühlewind et al. in [28] based on a modified version of aioquic, a QUIC and HTTP/3 python implementation supporting HTTP datagrams and CAPSULE frames for HTTP/3 and modifiable packet sizes and congestion control for QUIC. Experiments show a reduction in overhead and transmission time for increasing packet sizes which increases performance. Investigating the impact of RTTs and nested congestion control shows that with increased RTT, lower transmission times are generally observed in datagram mode compared to stream mode with the Reno CC algorithm showing better performance than Cubic or no CC. A significant improvement in transmission time is witnessed in stream mode for high loss rates with low delays on the link between the client and the proxy advocating the use of reliable streams in MASQUE proxying for lossy local links [28].

Discussion: The evaluation of MASQUE proxying suggests the potential loss recovery benefits of using stream mode for reliable data transmissions over wireless networks characterized by high non-congestive loss rates. Their work introduces the possibility of having simple link layer loss recovery mechanisms while offering explicit reliable data transfer with loss recovery when an application needs it. However, these promising results are limited by the used python based setup which is not optimized for performance. Further work using different stacks and emulation setups is required for better performance evaluation. Furthermore, security problems have to be considered when employing MASQUE proxying since it does not inspect the sent packets, therefore opening the possibility of malicious packets and replay attacks.

4.2. Combining Performance and Security

Motivation: GEO satellites suffer from long delays which negatively impact the *performance* and security of the satellite link. To deal with this issue, PEPs are employed by ISPs. The problem with PEPs is that they operate on clear data, thus raising security concerns for the satellite link. On the other hand, employing

 doi: 10.2313/NET-2024-04-1_15

approaches that support encryption, for instance VPN, is incompatible with PEP as the latter have to inspect clear TCP headers. The *trade-off* between performance and security is the crux of the problem in satellite communication motivating the development of QPEP by Pavur et al. [33]. QPEP is a new hybrid protocol between traditional PEP and VPN aiming to enhance the performance of secure satellite traffic making use of QUIC.

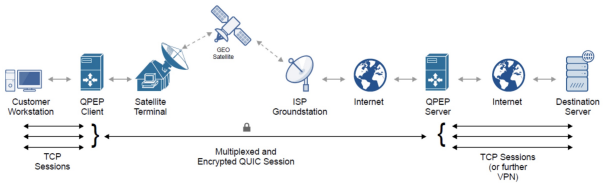


Figure 2: Simplified QPEP distributed architecture [33])

Implementation: Figure 2 illustrates the basic QPEP architecture. It is implemented as a transparent distributed PEP: The client establishes a persistent QUIC tunnel with the server for multiplexed and encrypted QUIC connection and maps incoming TCP connections to unique QUIC streams. It selectively terminates TCP connections discarding spurious ACKs and sends only relevant data converted to QUIC packets via the tunnel. We note that currently there is only an available TCP/IP stack implementation for tunneling [33].

Evaluation: QPEP was tested in simulations in [33] showing better performance than traditional PEPs. These promising results motivated the evaluation of its performance in a real world environment over commercially available satellite links by Huwyler et al. [34]. The metrics for measurements are goodput by measuring the download speed at the client side with varying payload sizes and Page Load Time (PLT) as an important metric for good user web experience.

An evaluation of different scenarios was conducted: *Plain connection* with neither performance enhancement nor security support, *PEPsal* as the only publicly available PEP, *OpenVPN* as an encrypted protocol and *QPEP*. Regarding goodput, QPEP outperformed all scenarios in a setup where the GEO provider proprietary PEP was not activated. Activating it did not enhance QPEP throughput in contrast to the other scenarios. Overall, QPEP attained higher goodput than OpenVPN especially in case of low payload sizes. The differences in PLTs between the different scenarios were less pronounced in the real testbed compared to the simulated one. QPEP outperformed OpenVPN in the setup with activated provider PEP improving PLT. Without it QPEP demonstrated the best performance among all protocols. Overall, QPEP proved its performance benefits over traditional PEPs and OpenVPN.

Discussion: The results obtained are limited by the "black box" nature of the provider networks causing certain ambiguities in the explanation of QPEP performance in specific scenarios. Further parameter tuning for the satellite link could help gain more insight into the performance benefits of QPEP [35]. Further work regarding the collaboration with different GEO providers and testing QPEP for LEO networks should be pursued.

4.3. Secure Middlebox Insertion in QUIC Connections

Motivation: QUIC with its end-to-end encryption opposes the idea of splitting the end-to-end connection to inspect or modify the exchanged information for performance enhancement purposes. A possible QUIC enhancement idea consists in the selective and controllable exposure of information to intermediary nodes in the network to allow the conscious insertion of middleboxes by the endpoints without violating the security of the communication. This idea is introduced by Kosek et al. as Secure Middlebox Assisted QUIC (SMAQ) [36].

Implementation and Evaluation of SMAQ: SMAQ inserts middleboxes that preserve the end-to-end security aspects while simultaneously being capable of adjusting certain functional aspects of QUIC to improve the performance. As illustrated in Figure 3 a *state handover mechanism* enables the endpoint to share its protocol state with an inserted middlebox by sharing the necessary keying material. This enables splitting of the end-to-end connection in two independent connections using an enhanced QUIC *connection migration mechanism* coupled with an added encryption layer during the QUIC handshake. SMAQ also supports the use of multiple middleboxes through transitive state handover. We direct the reader to [36] for a detailed design description.

In the same paper, a study case was conducted with SMAQ to use distributed PEPs in a QUIC satellite connection with both GEO and LEO satellite orbits. To evaluate the performance, a PEP optimized SMAQ using Hybla-Westwood [18] [37] as CC algorithm was compared to an end-to-end QUIC connection using NewReno w.r.t Bulk Download measured by the bytes received by the client over several time intervals and web performance measured by PLT. The results showed an overhead of a bit more than one RTT for SMAQ-PEP connection setup, an increase of bytes received with higher loss compared to normal QUIC and an overall better performance with higher RTT, loss rate, and byte transfer sizes.

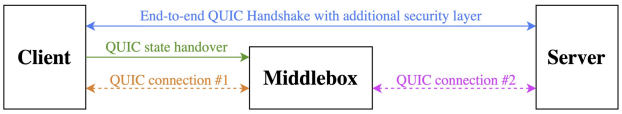


Figure 3: Overview of SMAQ design [36])

Discussion: Using SMAQ raises important security concerns. Its design is based on the assumption that middleboxes are trusted to manipulate and modify the connection violating the principles of privacy, integrity and authenticity. The current design only supports state handover started by the client and only during QUIC handshake which limits the potential performance benefits. Future work addressing these limitations should be pursued.

5. Conclusion and Future Work

Performance enhancing proxies are an important tool to enhance TCP performance in hybrid networks. In this survey we first investigate current PEP approaches for TCP. Although the number of open source PEP implementations is limited, we were able to find novel approaches in the current research. We investigate TCP performance enhancement over satellite networks presenting an older approach widely used in research. We discuss the idea of translating between different network architectures to enable the deployment of new architectures. We also introduce novel approaches for Multi-Domain Congestion Control. We then focus on exploring PEP possibilities for QUIC. We discuss interesting concepts such as explicit user-consent based proxying, combining the conflicting goals of performance and security in satellite networks, inserting middleboxes in a QUIC connection without violating the end-to-end security. Overall, the discussed ideas show promising performance gains for QUIC. However, they are limited by the used implementations that raise problems of scalability and security. Future work should further investigate these ideas while dealing with the mentioned limitations.

References

[1] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[2] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environments: problems and solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27–S32, 2005, publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1404595/

[3] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3135

[4] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. Los Angeles CA USA: ACM, Aug. 2017, pp. 183–196. [Online]. Available: https://dl.acm.org/doi/10.1145/3098822.3098842

[5] M. Kosek, H. Cech, V. Bajpai, and J. Ott, "Exploring Proxying QUIC and HTTP/3 for Satellite Communication," in *2022 IFIP Networking Conference (IFIP Networking)*. IEEE, 2022, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9829773/

[6] J. Pavur, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "A tale of sea and sky on the security of maritime VSAT communications," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1384–1400. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9152624/

[7] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: a new congestion control scheme for satellite IP networks," *IEEE/ACM Transactions on networking*, vol. 9, no. 3, pp. 307–321, 2001, publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/929853/

[8] R. Zullo, A. Pescapé, K. Edeline, and B. Donnet, "Hic sunt proxies: Unveiling proxy phenomena in mobile networks," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2019, pp. 227–232. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8784678/

[9] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tuxen, M. Welzl, D. Damjanovic, and S. Mangiante, "De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, p. 619, 2017. [Online]. Available: https://www.academia.edu/78032520/De_Ossifying_the_Internet_Transport_Layer_A_Survey_and_Future_Perspectives

[10] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. Berlin Germany: ACM, Nov. 2011, pp. 181–194. [Online]. Available: https://dl.acm.org/doi/10.1145/2068816.2068834

[11] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[12] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[13] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin, "Achieving faster access to satellite link bandwidth," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 4. IEEE, 2005, pp. 2870–2875. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1498578/

[14] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. Pittsburgh Pennsylvania USA: ACM, Aug. 2002, pp. 89–102. [Online]. Available: https://dl.acm.org/doi/10.1145/633025.633035

[15] P. Davern, N. Nashid, C. J. Sreenan, and A. Zahran, "HTTPEP: A HTTP performance enhancing proxy for satellite systems," *International Journal of Next-Generation Computing*, vol. 2, no. 3, pp. 242–256, 2011, publisher: Citeseer. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=137fd719d5aa77145b38397bbd5effb6de3ee1c9

[16] V. Farkas, B. Héder, and S. Nováczki, "A Split Connection TCP Proxy in LTE Networks," in *Information and Communication Technologies*, R. Szabó and A. Vidács, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7479, pp. 263–274, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32808-4_24

[17] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a Performance Enhancing Proxy for TCP satellite connections."

[18] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, Sep. 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/sat.799

[19] M. Handley, "Why the Internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Jul. 2006. [Online]. Available: http://link.springer.com/10.1007/s10550-006-0084-z

[20] K. Ciko, M. Welzl, and P. Teymoori, "PEP-DNA: A Performance Enhancing Proxy for Deploying Network Architectures," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, Nov. 2021, pp. 1–6, iSSN: 2643-3303. [Online]. Available: https://ieeexplore.ieee.org/document/9651953/

[21] D. John, "Patterns in network architecture: a return to fundamentals," 2007.

[22] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012, publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6231276/

[23] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe, "Open research issues in Internet congestion control," 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6077

[24] B. Trammell, M. Kühlewind, E. Gubser, and J. Hildebrand, "A new transport encapsulation for middlebox cooperation," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2015, pp. 187–192. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7390442/

[25] A. Mihály, S. Nádas, S. Molnár, Z. Krämer, R. Skog, and M. Ihlar, "Supporting multi-domain congestion control by a lightweight pep," in *2017 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*. IEEE, 2017, pp. 105–110. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8325922/

[26] Z. Kramer, S. Molnar, M. Pieska, and A. Mihaly, "A Lightweight Performance Enhancing Proxy for Evolved Protocols and Networks," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. Pisa, Italy: IEEE, Sep. 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9209304/

[27] Z. Krämer, M. Kühlewind, M. Ihlar, and A. Mihály, "Cooperative performance enhancement using QUIC tunneling in 5G cellular networks," in *Proceedings of the Applied Networking Research Workshop*. Virtual Event USA: ACM, Jul. 2021, pp. 49–51. [Online]. Available: https://dl.acm.org/doi/10.1145/3472305.3472320

[28] M. Kühlewind, M. Carlander-Reuterfelt, M. Ihlar, and M. Westerlund, *Evaluation of QUIC-based MASQUE proxying*, Dec. 2021, pages: 34.

[29] D. Perino, M. Varvello, and C. Soriente, "ProxyTorrent: Untangling the Free HTTP(S) Proxy Ecosystem," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. Lyon, France: ACM Press, 2018, pp. 197–206. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3178876.3186086

[30] D. Schinazi, "The CONNECT-UDP HTTP Method," Internet Engineering Task Force, Internet-Draft draft-ietf-masque-connect-udp-04, Jul. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-masque-connect-udp/04/

[31] D. Schinazi and L. Pardue, "Using Datagrams with HTTP," Internet Engineering Task Force, Internet-Draft draft-ietf-masque-h3-datagram-03, Jul. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-masque-h3-datagram/03/

[32] T. Pauly, E. Kinnear, and D. Schinazi, "An Unreliable Datagram Extension to QUIC," RFC 9221, Mar. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9221

[33] J. Pavur, M. Strohmeier, V. Lenders, and I. Martinovic, "QPEP: An Actionable Approach to Secure and Performant Broadband From Geostationary Orbit," in *Proceedings 2021 Network and Distributed System Security Symposium*. Virtual: Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_4A-1_24074_paper.pdf

[34] J. Huwyler, J. Pavur, G. Tresoldi, and M. Strohmeier, "QPEP in the Real World: A Testbed for Secure Satellite Communication Performance," in *Proceedings 2023 Workshop on Security of Space and Satellite Systems*. San Diego, CA, USA: Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2023/06/spacesec2023-239792-paper.pdf

[35] L. Thomas, E. Dubois, N. Kuhn, and E. Lochin, "Google QUIC performance over a public SATCOM access," *International Journal of Satellite Communications and Networking*, vol. 37, no. 6, pp. 601–611, Nov. 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/sat.1301

[36] M. Kosek, B. Spies, and J. Ott, "Secure Middlebox-Assisted QUIC," in *2023 IFIP Networking Conference (IFIP Networking)*. IEEE, 2023, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10186363/

[37] 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*. Rome Italy: ACM, Jul. 2001, pp. 287–297. [Online]. Available: https://dl.acm.org/doi/10.1145/381677.381704

The Path of a Packet Through the Linux Kernel

Alexander Stephan, Lars Wüstrich*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: alexander.stephan@tum.de, wuestrich@net.in.tum.de

***Abstract*—Networking stacks are the backbone of communication and information exchange. This paper investigates the TCP/IPv4 and UDP/IPv4 network stack of Linux, the most common server OS. We describe a trace of the most critical networking functions of the Linux kernel 5.10.8. Although Linux networking code documentation exists, it is often outdated or only covers specific aspects like the IP or TCP layer. We address this holistically, covering a packet's egress and ingress path through the Linux networking stack. Moreover, we highlight intricacies of the implementation and present how the Linux kernel realizes networking protocols. Our paper can serve as a basis for performance optimizations, security analysis, network observability, or debugging.**

***Index Terms*—linux kernel, network stack, packet processing**

1. Introduction

Nowadays, almost everything is networked, from a personal computer to a fridge [1]. Although networking is essential for modern computing, few know the complexity of getting a packet to and from a wire. Given the prevalence of Linux-based servers [2], [3], it is common for packets to traverse through the Linux network stack. However, understanding the intricacies of the complex packet processing within Linux takes time and effort. Nevertheless, this knowledge is often critical as it aids performance optimizations, security analysis, debugging, and network observability.

We base our investigation of the ingress and egress packet path on version 5.10.8 of the Linux kernel[1]. It is well-documented, stable, and contains modern features such as a Just-in-time (JIT) compiler for Berkely Packet Filters [4]. Primarily, we make observations on the kernel source code, which we link to referenced kernel symbols.

Although Linux kernel networking is becoming more diverse, e. g., with the addition of Multipath TCP [5], most traffic utilizes the standard TCP and UDP protocol stack. Moreover, despite the acceleration in IPv6 adoption, most devices still communicate over IPv4 [6]. Hence, we limit this analysis to TCP/IPv4 and UDP/IPv4.

The remainder of the paper has the following structure: Firstly, we compare this paper with existing literature in Section 2. Then, in Section 3, we explain the design of the general Linux networking stack and the `sk_buff` data structure. In Section 4, we inspect the intricacies of both the ingress and egress packet paths. Finally, in Section 5, we briefly summarize the most important findings.

1. https://elixir.bootlin.com/linux/v5.10.8/source

2. Related Work

We evaluated literature on the Linux network stack to the best of our knowledge. While doing so, we made the following observations.

Outdated Linux kernel versions. More elaborate papers emerged in the 2000s, using Linux kernel version 2 or 3 [7]–[9]. Although the implementation of older protocols in the network stack is stable, much time has passed. Therefore, we investigate possible deviations.

Fragmented Information. Many papers focus on specific layers, most commonly the TCP and IP implementation [10]–[13]. Others determine the causes of network overhead [14], [15]. A holistic view is lacking in those cases. In particular, even when authors describe the path of a packet throughout multiple layers [7]–[10], they omit UDP—in contrast to this paper.

Although there is a talk covering the whole ingress and egress path for Linux version 5 [16], it is high-level, mainly giving an intuition. Hence, we aim for a middle ground between detailed layer-specific information and high-level network stack tracing.

3. Background

We assume a basic familiarity with Linux and networking. However, we briefly describe essential Linux networking concepts relevant throughout the packet path.

3.1. Linux Networking Stack

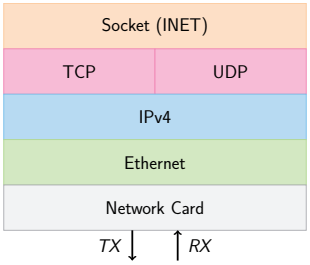


Figure 1: Depiction of the technologies used in the standard TCP/IP and UDP/IP stack in Linux, from user space to the wire.

As shown in Figure 1, a socket either passes a packet to the user space application or receives a packet from the implementation of the transport layer protocol, i. e., TCP or UDP. The IP layer then routes the packets to the network layer. Below this layer, Linux allows filtering

 doi: 10.2313/NET-2024-04-1_16

traffic via firewall rules. The network interface card (NIC) forwards the packets that it receives from the receive (RX) buffer to the kernel and transmits packets read from the transmit (TX) buffer.

3.2. Socket Buffers (`sk_buff`)

The kernel saves packets in C structures called `sk_buff`. Almost all functions along the packet path interact with it. `sk_buff` tracks packet metadata and maintains a start and end pointer to packet data in memory [17]. Using references to packet data allows for efficient packet modification by adjusting the pointers, e. g., when stripping a header away. Furthermore, `sk_buff` structures can be shared efficiently between different processes using memory references [17]. Consequently, cloning a packet is also efficient since only the metadata has to be copied [17], assuming a read-only workload. We show this in Figure 2. These properties of `sk_buff` form the basis of efficient packet processing on Linux.

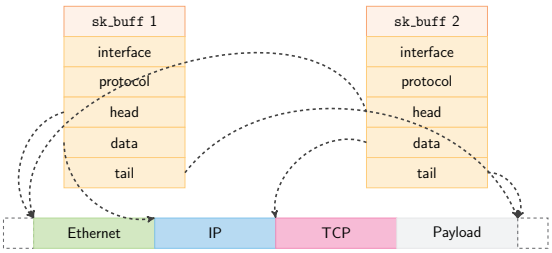


Figure 2: Two simplified `sk_buff` structures point to different locations within the same packet buffer. `head` marks the padded start of the buffer while `tail` points to the end of the actual packet data. `data` points to the currently processed header.

4. Packet Flow

Here, we are interested in both the ingress and egress paths. Both paths operate independently.

4.1. Egress Path

Firstly, we analyze the egress path, i. e., how Linux sends packets—from a user space application to the NIC as shown in Figure 3. Essentially, the egress side constructs the protocol headers, pushing them to `sk_buff` structures, which it sends out.

4.1.1. Socket Layer. All starts with a socket that has an associated domain, e. g., `AF_UNIX`, `AF_XDP`, or, in our case, `AF_INET` for IPv4. A system call wrapper function like `write()` or `sendto()` enables us to send data over the socket, e. g., as provided by the GNU C library [18]. In the context of this paper, we choose `write(filedescriptor, buffer, length)` to avoid unnecessary complexity. Writing to a file descriptor is a prime example of the UNIX philosophy *Everything is a file* since a file descriptor abstracts the socket [19].

For sockets, `write()` invokes the `sock_sendmsg()` function. It obtains the socket `struct sock` from the

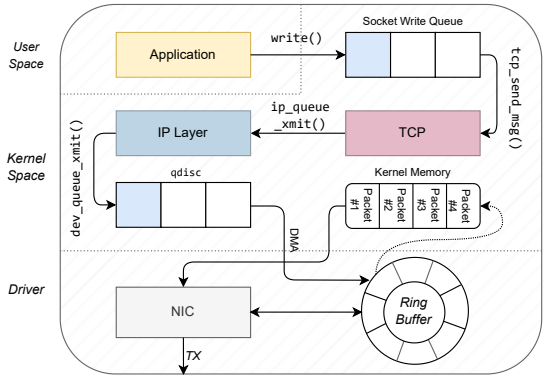


Figure 3: Egress path of a packet in case of TCP as described in Section 4.1 (adopted from [10]).

file descriptor provided by the user space application. Generally, sockets operate on socket control messages containing the process's Process ID (PID), User ID (UID), and Group ID (GID) [19]. `sock_sendmsg()` retrieves this control message from the `task_struct`, a Linux data structure that contains this information for the calling process. With this information, `sock_sendmsg()` typically passes the packet through Linux Security Modules (e. g., SELinux) to filter traffic.

Finally, it calls the corresponding transport layer handler, in our case TCP or UDP, via the macro `INDIRECT_CALL_INET`. The macro autonomously chooses the corresponding IPv4 or IPv6 variant of the transport protocol entry function, depending on the protocol specified in `sk_prot`, a field of the `sk_buff`.

4.1.2. Transport Layer. Here, we arrive at the IPv4 related entry functions, `tcp_sendmsg()` for TCP and `udp_sendmsg()` for UDP.

TCP. `tcp_sendmsg()` first waits for TCP connection establishment. Then, it allocates `sk_buff` structures for the segments and enqueues them to the socket write queue, as shown in Figure 3. `tcp_sendmsg()` also guarantees adherence to the Maximum Segment Size (MSS). After processing the queue, the kernel invokes `tcp_write_queue_tail()`. It also builds the TCP header and pushes the data from the user space into the `sk_buff`. If the data fits into the existing buffer, `skb_add_data_nocache()` is used. Otherwise, it creates new buffers, which is more expensive. It then sets the `transport_header` pointer to the beginning of this header. Next, it builds the network layer protocol header as specified in the socket options, e. g., IPv4 for `AF_INET`. `tcp_write_xmit()` guarantees that the kernel holds back data in case of congestion control restrictions. It also sets retransmission timers, i. e., resends the packet if it does not receive an `ACK` in time. Finally, `tcp_transmit_skb()` reads the write queue containing previously constructed segments and passes them to the network layer via the `queue_xmit()` function specified in the socket.

UDP. Similarly, there is `udp_sendmsg()`. Again, the function writes to the socket write queue. Next, the function waits until there are no pending frames for the UDP datagram. As before, the function builds the header, setting the destination port and the other fields.

 doi: 10.2313/NET-2024-04-1_16

There are corking and non-corking cases: corking describes waiting for frames to batch multiple UDP datagrams. Non-corking implies building `sk_buff` directly. After constructing the datagram, `ip_route_output_flow()` routes the packet and builds the network layer protocol header. Lastly, `ip_append_data()` creates an IP packet that combines multiple UDP datagrams. Overall, simplicity and absence of locking endorse that the UDP implementation is more performant than its TCP counterpart.

4.1.3. IP Layer. IP processing starts with the function `__ip_queue_xmit()`. Firstly, the function determines the route to the destination. If a route is already in `sk_buff->_skb_refdst`, it skips the routing process. In this case, the function builds the header immediately. However, if there is no destination, the routing process continues. It determines the destination from the socket field of the `sk_buff`, that, e. g., is set if the socket previously received an IP packet. If this is not possible, it queries the routing cache, called Forwarding Information Base (FIB)—a table that is generated from the IP routing table. Eventually, if there is still no route, it returns *host unreachable* and stops the processing. Otherwise, the kernel builds the IP header if it finds a route.

Now, `ip_options_build()` is called to set IP options. It marks the beginning of the header with the `network_header` field of the `sk_buff`. Next, it triggers the `LOCAL_OUT` stage of the Linux firewall mechanism `netfilter`. Afterward, `dst_output()` calls the actual routing function via a function pointer.

Then, the kernel calls the `ip_output()` routing function for the most common unicast packet. As the routing is complete, this stage is called `POST_ROUTING`. It updates the packet metadata and calls the `NF_INET_POST_ROUTING` hook. It sets `sk_buff` metadata and invokes netfilter once again. Furthermore, it fragments the packet if it exceeds the maximum length (Maximum Transmission Unit).

Then, after passing the packet through the `NF_INET_LOCAL_OUT` hook, `ip_output()` calls `ip_finish_output()`. It increments the counters for multicast and broadcast packets. It also checks that the `sk_buff` has enough space for the MAC header. The destination MAC address is either cached or determined by the neighbor output function `neigh_resolve_output()`. The latter utilizes the Address Resolution Protocol (ARP) [20]. In case there is no ARP reply, it queues the packet again. After obtaining the MAC address, the kernel constructs the Ethernet header, adding it to the `sk_buff`.

4.1.4. Ethernet Layer. Firstly, `dev_queue_xmit()` sets the `mac_header` field in the `sk_buff`, which is then passed to `tc_egress()`. It queues the packet in the queueing discipline (`qdisc`) [21]. As long as the NIC buffer is filled, `__qdisc_run()` dequeues the packets from the buffer. After some post-processing in `validate_xmit_skb()`, e. g., calculating the Ethernet checksum or adding VLAN tags, the kernel calls `ndo_start_xmit`, and consequently, adds the packet to the TX ring of the NIC. Eventually, the NIC's queue may be full. In this case, the kernel stops the `qdisc` [21] and queues `sk_buff`. Finally, it maps the packet to a fixed location in memory for Direct Memory Access (DMA) after adding more `sk_buff` metadata. `dev_direct_xmit` allows circumventing the `qdisc` [21], directly writing the packet to the TX ring of the NIC. eXpress Data Path (XDP) [22] is a use case of this. Eventually, the function notifies the NIC via an interrupt to end the processing and frees the `sk_buff`.

4.2. Ingress Path

Now, we trace the path of a packet that arrives at the NIC until a user space application reads it through a socket, see Figure 4. Most notably, it analyzes the headers to determine the following function call and strips them.

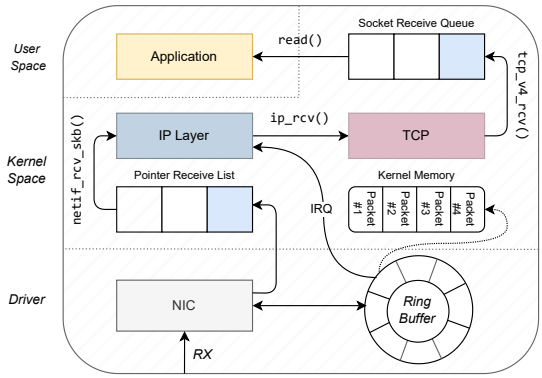


Figure 4: Ingress path of a packet in case of TCP as described in Section 4.2 (adopted from [10]).

4.2.1. Ethernet Layer. After verifying and optionally zeroing the Ethernet checksum and applying a MAC address filter, the NIC copies the packet to the system's memory via DMA. Then, it notifies the operating system via an interrupt and indicates the location of the packet data. With this, the operating system can allocate an `sk_buff`. Now, the kernel inserts metadata into the `sk_buff`, like the `protocol` field (Ethernet), the receiving `interface`, and the `packet` type, in our case, IP.

At this stage, the kernel knows the start of the Ethernet header, so it sets the `mac_header` field to the beginning of the `sk_buff`. Finally, it removes the Ethernet header from the `sk_buff` before it passes it further up the network stack. Next, the packet arrives in `netif_receive_skb()`. The function clones `sk_buff` and forwards it to the virtual TAP interface. The TAP interface enables communication between Virtual Machines (VMs) and the host within the same network. Another important case here is forwarding VLAN-tagged packets to the VLAN interface. Furthermore, when the interface has a physical master, i. e., it is a virtual interface or part of a network bridge, `rx_handler()` steals the packet. `rx_handler()` also sets the `network_header` field of the `sk_buff`. Finally, it calls the IPv4 protocol handler function `ip_rcv()`.

4.2.2. IP Layer. The Ethernet layer passes the packet to the IP layer via the function `ip_rcv()`. Again, `ip_rcv()` inspects the MAC address and drops foreign ones. Then, the version, length, and checksum fields are verified. Next, the function sets the `transport_header` field of the `sk_buff`. It also applies `netfilter`'s `PRE_ROUTING` rule. It implements the filter by forwarding the packet to the `NF_INET_PRE_ROUTING` hook. The hook gets a

 doi: 10.2313/NET-2024-04-1_16

pointer to the ip_rcv_finish() function that it calls after completion. If a network layer master device is registered, it passes the sk_buff to its handler. It calls ip_route_input_noref(), which reads the IP header from the sk_buff. Next, the kernel processes IP options via ip_rcv_options(). Afterward, it calls the previously selected routing function via dst_input(). There are three options for routing a packet:

1) ip_forward: This function activates for packets not addressed to the current machine. It proceeds by forwarding the packet without additional processing.
2) ip_local_deliver(): If we are the final receiver of the packet (*localhost*), the kernel does not forward the packet but passes it up the networking stack.
3) ip_mr_input(): This function is for multicast packets, i. e., addressed to a multicast address.

As we are mainly interested in how a packet is handled at the final receiver, taking all layers into account, we continue with ip_local_deliver(). Most importantly, this function takes care of IP fragmentation by calling ip_defrag(), queueing packets until receiving all fragments. Afterward, the event NF_INET_LOCAL_IN triggers, which in return calls ip_local_deliver_finish(), stripping the IP header from the sk_buff. Finally, it passes the packet from the IP to the TCP/UDP layer via the dst_input() function to the tcp_v4_rcv() or function. It determines the corresponding protocol handler by inspecting the header pointing to the sk_buff.

4.2.3. Transport Layer. Now, we inspect the counterpart of the egress TCP and UDP functions.

TCP. First, the segment arrives at the transport layer function tcp_ipv4_recv() with the sk_buff header pointer moved to the start of the TCP or UDP header. Then, it validates the transport header via pskb_may_pull(), validating the TCP checksum. As before, it removes the TCP header from the sk_buff. To pass the packet further, it locates the corresponding TCP socket via __inet_lookup_skb(). It writes the packet to the socket receive queue (see Figure 4) and signals that new data is available, e. g., via SIGIO or SIGURG. This notification mechanism allows for efficient polling of sockets. As for the egress, the kernel maintains the TCP state machine during packet processing. e. g., it processes no new packets for TCP connections terminated via a TCP_CLOSING.

We briefly highlight two important cases during processing: TCP_NEW_SYN_RECV and TCP_TIME_WAIT. TCP_NEW_SYN_RECV means that there is a new connection. In this case, the kernel refuses the connection at TCP level via tcp_filter(). During TCP_TIME_WAIT, the kernel discards any further TCP segments.

Furthermore, there is a slow and a fast path. The slow path contains more error checks and lookups. In contrast, the fast path is optimized for speed, not allowing introspection and traffic analysis. With the slow path, we wait until the state machine is at TCP_ESTABLISHED in tcp_v4_do_rcv(). Once updated, tcp_v4_do_rcv() calls tcp_rcv_established(), which processes packets both in the fast and slow paths. It also validates that sequence numbers are ascending. The fast path copies the packet directly to the user space. The kernel always tries to use the fast path, if possible. But when, e. g., establishing a TCP connection, this not possible since the kernel has to track the new connection.

After handling the TCP state machine and choosing the path, the kernel enqueues the packet into the socket queue so the user program can read it (see Figure 4).

Since TCP is very complex, covering further aspects is beyond this paper's scope. However, [7], [10], [11] describe it in more detail.

UDP. Compared to TCP, the implementation of UDP is less complex. It starts with udp_rcvmsg() called via dst_input() in the IP layer. First, the function calls __skb_recv_udp() to read the datagram from the socket with a previously calculated offset. In particular, it continuously tries to read a sk_buff from the socket, eventually stopping when a new UDP datagram arrives. The checksum of the datagram is then validated. Then, the function copies the destination IP and UDP port to map the datagram to the correct socket. Consequently, it consumes the UDP datagram via skb_consume_udp(). Finally, it adjusts the peek offset, handles reference counters, and frees the sk_buff via __consume_stateless_skb().

4.2.4. Socket Layer. Here, the kernel collects the new data written to a TCP or UDP socket via the read() function from a socket, dequeuing the packet from the socket receive queue (see Figure 4). To match the use of IPv4 in the egress, we use an AF_INET receiving socket. The function sys_recv() enables this, first calling sys_recvfrom() to look up the socket. Then, it calls sock_recvmsg() to read from the socket and passes the received message through Linux Security Modules, similar to the egress. For IPv4, inet_recvmsg() calls either tcp_recvmsg() or udp_recvmsg(). They dequeue the packet's content and write it to a userspace buffer, e. g., an array on the heap. Finally, they free the sk_buff.

5. Conclusion

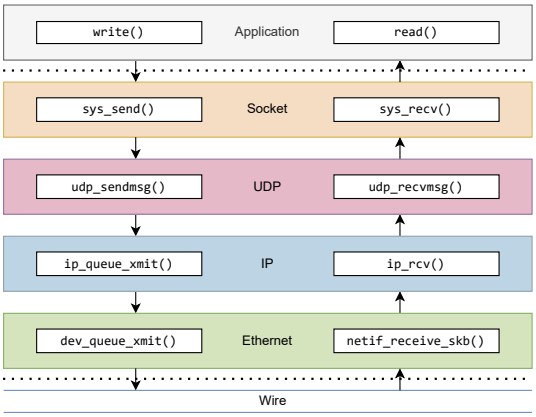


Figure 5: An overview of the most important functions in both egress and ingress for UDP, as described in Section 4.

This paper presented how a packet traverses the Linux kernel for TCP/IPv4 and UDP/IPv4. Figure 5 illustrates a recap of the packet egress and ingress path, highlighting the most important functions of each layer. Moreover, we described the intricacies of packet processing, including routing, filtering, and queuing mechanisms employed by

the Linux kernel. Furthermore, we have seen how the different layers in the kernels communicate. By leveraging this knowledge, network administrators and developers can make informed decisions when optimizing network performance, designing security measures, or troubleshooting networking issues.

Overall, the observed changes to the existing literature are primarily enhancements rather than rewrites, e. g., refactorings or security improvements. A prime example is the choice of initial sequence numbers for TCP. For security reasons, the kernel authors revised the underlying hash algorithm multiple times [23]. The conservative changes make sense, as the protocols remain mostly untouched while the impact of errors is high. Performing a similar analysis for Multipath TCP or QUIC is future work.

References

[1] T.-H. Lee, S.-W. Kang, T. Kim, J.-S. Kim, and H.-J. Lee, "Smart Refrigerator Inventory Management Using Convolutional Neural Networks," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[2] W3Techs. (2019) Usage Statistics of Operating Systems for Websites. https://w3techs.com/technologies/overview/operating_system. [Online; accessed 02-December-2023].

[3] ——. (2019) Usage Statistics of Unix for Websites. https://w3techs.com/technologies/details/os-unix. [Online; accessed 02-December-2023].

[4] J. Corbet. (2011) A JIT for Packet Filters. https://lwn.net/Articles/437981/. [Online; accessed 02-December-2023].

[5] C. Paasch and O. Bonaventure, "Multipath TCP," *Commun. ACM*, vol. 57, no. 4, p. 51–57, apr 2014. [Online]. Available: https://doi.org/10.1145/2578901

[6] M. T. Hossain, "A Review on IPv4 and IPv6: A Comprehensive Survey," 01 2022, International Interconnect Technology Conference (IITC). [Online]. Available: https://doi.org/10.13140/RG.2.2.18673.61284

[7] J. Crowcroft and I. Phillips, *TCP/IP and Linux Protocol Implementation: Systems Code for the Linux Internet*. USA: John Wiley & Sons, Inc., 2001.

[8] A. Chimata, "Path of a Packet in the Linux Kernel Stack," 01 2005. [Online]. Available: https://www.cs.dartmouth.edu/~sergey/netreads/path-of-packet/Network_stack.pdf

[9] C. Guo and Z. Shaoren, "Analysis and Evaluation of the TCP/IP Protocol Stack of Linux," vol. 1, 02 2000, pp. 444–453 vol.1.

[10] Helali Bhuiyan and Mark E. McGinley and Tao Li and Malathi Veeraraghavan, "TCP Implementation in Linux : A Brief Tutorial," 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:14676835

[11] Antti Jaakkola, "Implementation of Transmission Control Protocol in Linux," 2012. [Online]. Available: https://wiki.aalto.fi/download/attachments/70789052/linux-tcp-review.pdf

[12] F. U. Khattak. (2012) IP Layer Implementation of Linux Kernel Stack. [Online]. Available: https://wiki.aalto.fi/download/attachments/70789059/linux-kernel-ip.pdf

[13] M. C. Wenji Wu, "The Performance Analysis of Linux Networking - Packet Receiving," 2006, 15th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2006).

[14] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding Host Network Stack Overheads," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 65–77. [Online]. Available: https://doi.org/10.1145/3452296.3472888

[15] M. Abranches, O. Michel, and E. Keller, "Getting Back What Was Lost in the Era of High-Speed Software Packet Processing," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, ser. HotNets '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 228–234. [Online]. Available: https://doi.org/10.1145/3563766.3564114

[16] J. Benc, "The Network Packet's Diary: A Kernel Journey," 2018, devcon.cz 2018.

[17] The kernel development community. struct sk_buff. https://docs.kernel.org/networking/skbuff.html. [Online; accessed 02-December-2023].

[18] G. Foundation. (2023) The GNU C Library Reference Manual. https://sourceware.org/glibc/manual/2.38/html_mono/libc.html#Transferring-Data. [Online; accessed 05-December-2023].

[19] M. Kerrisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*, 1st ed. USA: No Starch Press, 2010.

[20] "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826, Nov. 1982. [Online]. Available: https://www.rfc-editor.org/info/rfc826

[21] *tc - traffic control utility*, Linux Documentation Project, December 2023, https://man7.org/linux/man-pages/man8/tc.8.html.

[22] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 54–66. [Online]. Available: https://doi.org/10.1145/3281411.3281443

[23] F. Gont and S. Bellovin, "Defending Against Sequence Number Attacks," RFC 6528, Feb. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6528

ISBN 978-3-937201-79-5
DOI 10.2313/NET-2024-04-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)