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)

Summer Semester 2023

March 6, 2023 – August 17, 2023

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)

Summer Semester 2023

Munich, March 6, 2023 – August 17, 2023

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

Network Architectures
and Services
NET 2023-11-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Summer 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 SS 23
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, March 6, 2023 – August 17, 2023
ISBN: 978-3-937201-78-8

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2023-11-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2023-11-1
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 Summer 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 Thomas Dietrich with the paper *Common Workflow Language Execution on the I8-Testbed* and Ulkar Aslanova with the paper *Wireless Time Synchronization in IEEE 802.11* .

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, December 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

Philippe Buschmann (phil.buschmann@tum.de)
Technical University of Munich

Sebastian Gallenmüller (gallenmu@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

Filip Rezabek (rezabek@net.in.tum.de)
Technical University of Munich

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

Christoph Schwarzenberg (schwarzenberg@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

Henning Stubbe (stubbe@net.in.tum.de)
Technical University of Munich

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

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

Richard von Seck (seck@net.in.tum.de)
Technical University of Munich

Seminar Homepage

`https://net.in.tum.de/teaching/ss23/seminars/`

Contents

Block Seminar

Seminar

MsQuic – A High-speed QUIC Implementation

Manuel Bünstorf, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: m.buenstorf@tum.de, jaeger@net.in.tum.de

***Abstract*—The QUIC protocol is designed as a more flexible alternative for the TCP / TLS stack and is implemented in user-space. Thus several implementations exist, each with its own strengths and weaknesses. In this paper, we take a close look at MsQuic, a high-speed QUIC implementation by Microsoft. We give an overview of the library, showcase its strengths, and investigate in what projects it is already deployed. Also, we evaluate MsQuic regarding performance on different hardware architectures. Comparing the goodput of MsQuic with other QUIC implementations, we found that MsQuic outperforms them all, with the largest difference of goodput on older hardware.**

***Index Terms*—quic, msquic, goodput, performance, high-speed, transport, measurements**

1. Introduction

QUIC is a new transport protocol designed to replace the existing TCP / TLS protocol stack. It was originally developed by Google [1] and is now standardized by the Internet Engineering Task Force [2]. It improves performance and latency, by eliminating redundant handshakes, streamlining the handshake procedure. It is built on top of UDP and runs in user-space. Many different QUIC implementations exist as a result of being built in user-space, each having its strengths and weaknesses. They differ in their implementation language and in their design. This includes congestion and flow control, packet size, retransmission handling, and more. One of these implementations is MsQuic by Microsoft [3]. It is open source and designed to be a high-performance, general purpose and cross-platform implementation of the QUIC protocol. MsQuic is deployed in Windows [4] and other applications. In this paper, we take a look at QUIC, how it works on a basic level, and what improvements it makes over the existing TCP and TLS networking stack. We look at MsQuic regarding the team behind it, its goals and design as well as the process behind developing a QUIC library with a focus on performance. We showcase the high-level design of the library and shortly introduce its API. We also show results of our measurements on MsQuic and competing implementations, testing their claims of being optimized for maximal throughput.

The remaining part of the paper is structured as follows. Firstly, we introduce QUIC and explain important parts of its design. Then we showcase MsQuic, talking about its goals, high-level architecture and its development. Following this, we present our measurements regarding MsQuic and competing implementations. Finally, we summarize related work that gives more context to the performance measurements, before we conclude the paper.

2. Background

QUIC is intended as a next-generation transport protocol that powers the new HTTP / 3 [5]. It replaces TCP / TLS, taking care of flow control. It also handles stream multiplexing, which was previously handled by the HTTP layer. QUIC was originally developed by Google as a successor to SPDY, their first attempt at designing a next-generation transport protocol. Taking the lessons learned from SPDY, they started development on QUIC, which was standardized in 2021 by the IETF as RFC 9000 [2].

2.1. Rapid deployment

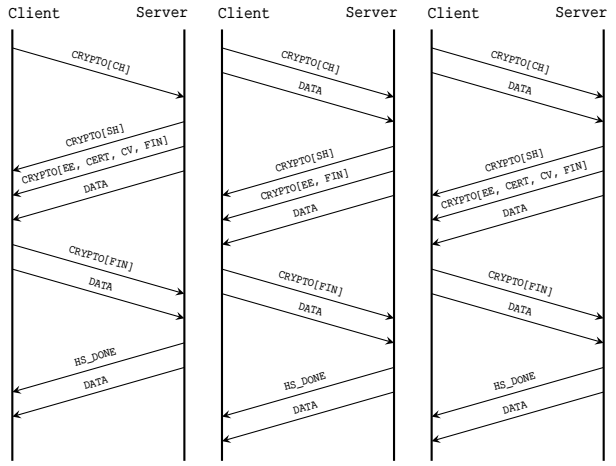
One of the main goals when designing QUIC was to make rapid deployment possible, which also fostered the general development process by Google. To achieve this they made two key choices for the design of the protocol [1]. The first one was to build QUIC in user-space rather than kernel-space, allowing deployment as part of applications like web browsers[1], in contrast to slow-moving kernels of operating systems. The other decision was to use the existing UDP protocol as a lightweight layer underneath QUIC, leveraging existing network infrastructure that already supports UDP traffic. That eliminated the need to make any changes to middleboxes for them to support the new protocol. These two decisions made it possible for Google to roll QUIC out to users as part of their products very quickly. It also gave them the opportunity to iterate on their designs in the real world, having the tools to A / B test changes with real-time feedback and monitoring.

2.2. Performance improvements

Another goal of QUIC was to reduce handshake latency and enhance performance over the existing TCP / TLS stack. First and foremost by reducing the amount of network round trips that are required for a TCP and a TLS handshake. It takes three round trips for a TCP / TLS handshake between a client and server before the client can send the actual request to the server. QUIC combines the handshakes for the transport layer

1. Google used their browser Chrome to roll out the QUIC protocol to their users.

 doi: 10.2313/NET-2023-11-1_01



(a) 1-RTT (b) 0-RTT (c) Rejected 0-RTT

Figure 1: QUIC handshake timelines

and encryption into one. With that alone, QUIC is able to alleviate the overhead needed for setting up a connection [1].

Establishing a connection between a client and server with QUIC is done in two different ways, as shown in Figure 1. An initial connection of a client that has not connected to the server before is depicted in Figure 1a. The client sends an incomplete `Client Hello` to the server, prompting a `Server Hello` from the server, together with cryptographic data like the certificate, which the client needs to establish an encrypted and trustable connection. At this point the server can already send encrypted `DATA` to the client. With the cryptographic data from the server, the client sends a complete `Client Hello` and encrypted `DATA` can be sent. The server responds with a `Handshake Done` message, indicating that a secure connection was established, and from there the connection to the client is established. This results in only one network round trip (1 RTT) of delay to establish a new connection, only a third of the overhead TCP/TLS has. After this initial connection, the client can cache the server configuration as well as the cryptographic data to use in later connections with the same server. As illustrated in Figure 1b, the client can avoid sending an incomplete `Client Hello` and is able to use the cached data to immediately send a complete `Client Hello` as well as `DATA`, allowing for a connection without any overhead. Lastly, the data cached by the client might become invalid after some time, for example because the server changed its configuration. When the client tries to initiate a new connection with an invalid encryption, the server will respond with the new cryptographic data. At this point the client will proceed with the regular 1-RTT procedure for connection establishment as seen in Figure 1c.

3. MsQuic

MsQuic is the open source implementation of the QUIC protocol by Microsoft [3]. Its initial release was on the 27th of November 2019. It was developed by the datapath and transports team at Microsoft, headed by Nicholas Banks [6]. He is the primary QUIC architect and developer and is responsible for the project. Later, the team was split into a dedicated datapath team and transports team, the latter of which now continues the development of the library. The code for MsQuic was open sourced on the 28th of April 2020 on GitHub and is licensed under a MIT License [3] that provides complete freedom over the code, allowing unconfined commercial use, modification and distribution. This, however, is without any liability or warranty.

Commercial software exists that already uses MsQuic. For example, it is already used as the kernel mode *msquic.sys* in Windows 11 and Windows Server 2022 and is used by the Windows kernel mode *http.sys* for the purpose of providing HTTP/3 capabilities [7]. The SMB protocol offers the usage of QUIC as an alternative to TCP, by using MsQuic under the hood [8]. The .NET Core ecosystem uses MsQuic to provide QUIC functionality as well [4]. The Microsoft Game Development Kit, which provides the possibility of using QUIC as a transport protocol to implement latency critical network code in games, also using MsQuic as the implementation of choice [9].

3.1. Goals

When Banks and his team started development on MsQuic their main focus was to build a general purpose QUIC implementation that can be used by other products from Microsoft to leverage QUIC as a next-generation transport protocol [4]. The two main components that wanted to use MsQuic right from the beginning were the HTTP kernel mode of Windows and the SMB protocol, both of which run in the Windows kernel. This meant that MsQuic needed to run in the kernel as well, in order to be used by both HTTP and SMB in Windows. A module inside of the Windows kernel has a few constraints, primarily regarding the implementation language. Kernel code for Windows has to be written in C or in a restricted form of C++. The other constraint is having to implement an asynchronous network IO model.

Although both HTTP and SMB were two main use cases for MsQuic from the very beginning, they had different requirements [4]. HTTP requires a high number of requests per second as well as low latency for connections. SMB on the other hand benefits from high throughput on a single connection. This meant MsQuic needed to be optimized for both cases, handling as many requests per second as possible while still enabling bulk transfer on single connections. A bit later into development .NET core team at Microsoft was interested in using their in-house QUIC implementation to provide QUIC functionality for the .NET ecosystem. Since .NET is a cross platform framework, this meant that the goals for MsQuic now also included providing cross platform support, for both Windows and Linux as well as Xbox.

3.2. Design

On a high level, MsQuic is split into two parts [4]. The core protocol that implements all of the platform independent protocol logic, as well as a platform specific abstraction layer that handles the interaction with constructs tied to the OS like sockets and threads. An

 doi: 10.2313/NET-2023-11-1_01

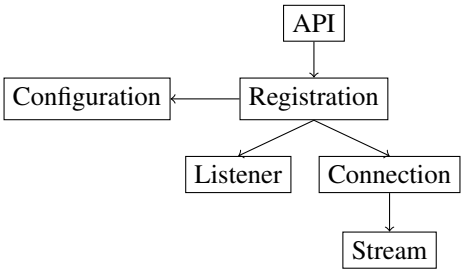


Figure 2: Object model of the API

asynchronous IO model is used for the data path, utilizing asynchronous calls into lower layers to send data and using upcalls back to the higher layer to complete these sends. To improve performance IO operations are batched between the Application / MsQuic layers and between the MsQuic / UDP layers. This batching works together with UDP send segmentation and receive coalescing [4].

The threading model of the library is comprised of two different types of threads, data path threads and core threads [4]. Data path threads handle UDP receive calls and basic QUIC validation, before queuing each packet to its associated connection object. The interaction of the data path threads with the UDP layer is platform dependent. Core threads process the queues of every connection object, performing the majority of processing. This division of the handlers for packet receives and the packet processing allows for a practically lock free execution of the code. It also makes horizontal and independent scalability possible, where both a high amount of requests and bulk throughput can be prioritized individually.

3.3. Prioritizing performance

One of the primary concerns when developing MsQuic is performance [3]. As a general purpose implementation, however, performance can mean different things. For HTTP, a high number of requests per second is crucial, while for file sharing high throughput is important. In order to provide performance in all scenarios, they have worked to standardize a measure for performance and created a process for testing the performance of their QUIC implementation in different scenarios [10]. These measures consist of single connection upload and download speeds, requests per second and handshakes per second. Testing for these scenarios is done automatically for every merge and pull request to the main branch. This provides real time feedback of changes in performance that can be tracked back to a certain commit. Thresholds are in place to automatically reject any pull requests that do not meet specified performance requirements. All of these metrics are publicly available on the MsQuic performance dashboard [11]. This commitment by the MsQuic team to keep track of performance changes on every commit has helped them a lot to achieve their goal of building a high-performance QUIC implementation.

3.4. API overview

The library exposes an API providing different objects encapsulating specific parts of MsQuic handling certain functionalities [12]. An overview of these objects is shown in Figure 2. The *Registration* object provides an execution context for the child objects; typically only one *Registration* per application should be created. A *Regestration* is associated with a *Configuration* object abstracting all of the settings available to change the behavior of the library. When building a server *Listener* objects are used to accept incoming connections from clients. The more *Listener* objects are created, the more simultaneous requests the server will be able to handle. A successfully accepted connection by a *Listener* will result in a new *Connection* object representing the connection between the server and a client. This *Connection* object can create or accept arbitrarily many *Stream* objects, each one representing an individual QUIC stream that is used for data transmission.

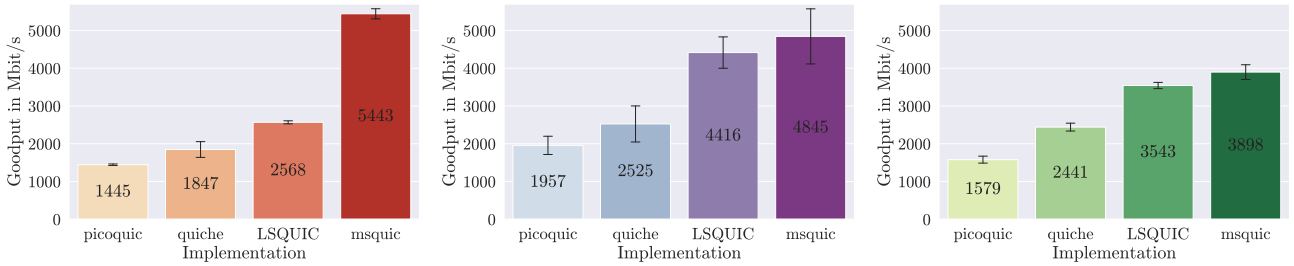
The API uses callbacks to indicate that an event has happened. The functions that are registered to these callbacks are not run in separate threads to MsQuic and as such should keep the execution time to a minimum, to not block execution of the library.

4. Evaluation

We compare the performance of MsQuic to other QUIC implementations and chose the following three. Quiche, which is the QUIC implementation from Cloudflare written in Rust. LSQUIC, an implementation from LiteSpeed, which is written in C and lastly picoquic, that has the goal of being a minimal but fully functional implementation of the QUIC protocol and is also written in C. We chose these three implementations because they prioritize performance and most importantly are all written in a systems programming language, making them good targets for performance and speed comparisons with MsQuic.

4.1. Setup

To conduct our measurements we used the test framework by Jaeger et al. [13], which allowed us to execute the tests on real hardware inside of a testbed. With this framework the measurement procedure was the following. Two seperate hosts are selected, one acting as a client and one as a server. The server hosts a webserver, exposing a 1073 MB large file and the client attempts to download it. Measuring the time this data-transfer takes, we calculate the average transmission speed. After the transfer is complete we make sure that the file was transmitted properly and did not get altered during transfer. This is called goodput instead of throughput because instead of measuring the raw amount of data transmitted, it takes into account what portion of the data is actually useful. It gets impacted negatively by retransmissions of packets and does not count packet headers as useful data and because of that it is a better representation of real world usage. We conducted these measurements ten times for each implementation, to reduce random deviations in the measured goodput. We performed this procedure on three different host pairs, with differing CPUs, to examine the how the performance of the QUIC implementations changes with different hardware. Each of the first two hostpairs had an Intel Xeon E5-1650 v3 CPU with 6 cores and 12 threads, running at 3.5Ghz. This is an older CPU



(a) Intel Xeon E5-1650 v3 (b) Intel Xeon Gold 6312U (c) AMD EPYC 7543

Figure 3: Goodput of the four QUIC implementations on different CPU architectures

from around 2014, giving insight into the performance on an aged Haswell architecture. The other two hostpairs we tested on had fairly recent CPUs: Intel Xeon Gold 6312U with 24 cores and 48 threads running at 2.4Ghz and the second hostpair used AMD EPYC 7543 CPUs with 32 cores and 64 threads, running at 3.7Ghz. The connection between each of the hostpairs was made with a 10GBase-T Cat6a network cable with full duplex mode.

4.2. Results

Our results are visualized in Figure 3, with the black lines at the top of the bars indicating the standard deviation of the measured samples. Interestingly MsQuic performed best on the older hardware, as seen in Figure 3a, achieving a mean goodput of 5443 $\mathrm{Mbit\,s^{-1}}$ and significantly outperforming all other implementations. On the newer Intel CPU all implementations achieve a higher goodput, except for MsQuic, which loses around 598 $\mathrm{Mbit\,s^{-1}}$ of mean goodput as depicted in Figure 3b. To be noted as well is the larger deviation of the measurements, indicating that the speed of the connection was unstable. This is not the case for the AMD host pair as illustrated in Figure 3c. None of the implementations achieve as high goodputs as measured on the new Intel CPU, but the achieved speeds are much more stable. MsQuic achieved the lowest goodput on this host pair, with a mean of only 3898 $\mathrm{Mbit\,s^{-1}}$.

From our resulst it seems like MsQuic achieves higher goodputs on older hardware, outperforming all the other implementations by a large margin. MsQuic still surpasses all the other implementations on the newer hardware, however with a much smaller margin. This behaviour might be caused by differences of the CPU architectures, but more measurements and research needs to be done to reach a conclusion.

5. Related work

This paper takes a close look at MsQuic, measuring its performance and comparing it to other QUIC implementations. A number of similar works exist that investigate the improvements the QUIC protocol makes [14]–[16]. Carlucci et al. check if QUIC can be deployed safely and compare QUIC to HTTP and SPDY [15]. The work of Cook et al. discovered specific use cases and conditions where QUIC is of high interest [14]. Yu et al. measured the performance of QUIC in production environments, locating the largest bottlenecks of deployed QUIC applications [16]. Seemann and Iyengar introduce the QuicInteropRunner, a framework to test interoperability and performance between different QUIC implementations [17].

Other similar works propose concrete techniques to improve the performance of QUIC implementations [18]–[20]. Yang et al. tackle the problem of QUIC using up to 3.5 times the CPU cycles of optimized TCP and TLS implementations [18]. They examined QUIC implementations to discover the computationally most intensive parts and using their findings to define an architecture for offloading these calculations to NICs. Tyunyayev et al. introduce picoquic-dpdk, a modification of picoquic that uses the DPDK library to bypass the Linux kernel networking stack, reducing the amount of slow context switches [19]. A different approach to circumventing large amounts of context switching is presented by Wang et al. [20]. They developed an implementation of QUIC that runs in kernel-space and used it to more accurately compare TCP and QUIC.

6. Conclusion

The last few years have been a success story for QUIC. It provides significant improvements over the existing TCP / TLS stack. Deployment of this new technology was fast and user adoption was quick. All because QUIC is built on top of UDP, making it effortless to deploy in existing network infrastructure and because it runs in user-space, making it not depend on operating systems to implement it. Between the many implementations of the QUIC protocol MsQuic stands out with its great performance, not only in regards to raw goodput but also the ability to handle a large number of requests per second. The team building MsQuic at Microsoft, has shown their dedication to building a high performance QUIC implementation by putting great efforts into automatic performance measurements and testing. Ensuring that every change to the code base is inline with their performance goals. On top of that MsQuic is open source under an MIT License, has cross platform support and good documentation and is under active development backed by a major corporation. We were able to validate their claims of high performance. MsQuic has double the goodput of LSQUIC, the second fastest QUIC implementation we tested. All of these points make MsQuic a good choice for any project that needs support for the QUIC protocol.

References

[1] 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*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[2] 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

[3] Microsoft, "MsQuic," 2023, last accessed 24 March 2023. [Online]. Available: https://github.com/microsoft/msquic

[4] N. Banks, "MsQuic - QUIC Performance Talk," 2021, last accessed 26 March 2023. [Online]. Available: https://www.youtube.com/watch?v=Icskyw17Dgw

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

[6] "Nicholas Banks LinkedIn," last accessed 13 May 2023. [Online]. Available: https://www.linkedin.com/in/nicholas-banks-a3977520/

[7] Microsoft, "Platform Support of MsQuic," last accessed 8 May 2023. [Online]. Available: https://github.com/microsoft/msquic/blob/9f74f69d0c16fadb62a332246daabac704bc7db0/docs/Platforms.md

[8] ——, "SMB over QUIC," last accessed 8 May 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/storage/file-server/smb-over-quic

[9] ——, "MsQuic for Microsoft Game Development Kit," last accessed 8 May 2023. [Online]. Available: https://learn.microsoft.com/en-us/gaming/gdk/_content/gc/networking/overviews/game-mesh/msquic-intro-networking

[10] N. Banks, "QUIC Performance," 2021, last accessed 26 March 2023. [Online]. Available: https://datatracker.ietf.org/doc/draft-banks-quic-performance

[11] ——, "MsQuic Performance Dashboard," last accessed 26 March 2023. [Online]. Available: https://microsoft.github.io/msquic/

[12] Microsoft, "MsQuic API," 2023, last accessed 24 March 2023. [Online]. Available: https://github.com/microsoft/msquic/blob/main/docs/API.md

[13] B. Jaeger, J. Zirngibl, M. Kempf, K. Ploch, and G. Carle, "QUIC on the highway: Evaluating performance on High-Rate links," in *International Federation for Information Processing (IFIP) Networking 2023 Conference (IFIP Networking 2023)*, Barcelona, Spain, Jun. 2023.

[14] 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.

[15] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: An Experimental Investigation of QUIC," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 609–614. [Online]. Available: https://doi.org/10.1145/2695664.2695706

[16] A. Yu and T. A. Benson, "Dissecting Performance of Production QUIC," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1157–1168. [Online]. Available: https://doi.org/10.1145/3442381.3450103

[17] M. Seemann and J. Iyengar, "Automating QUIC Interoperability Testing," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 8–13. [Online]. Available: https://doi.org/10.1145/3405796.3405826

[18] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC Quicker With NIC Offload," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–27. [Online]. Available: https://doi.org/10.1145/3405796.3405827

[19] N. Tyunyayev, M. Piraux, O. Bonaventure, and T. Barbette, "A High-Speed QUIC Implementation," in *Proceedings of the 3rd International CoNEXT Student Workshop*, ser. CoNEXT-SW '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 20–22. [Online]. Available: https://doi.org/10.1145/3565477.3569154

[20] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 227–234. [Online]. Available: https://doi.org/10.1145/3242102.3242106

A Scheme Towards Reproducibility

Felix Christ, Henning Stubbe*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: felix.christ@tum.de, stubbe@net.in.tum.de

***Abstract*—The result of compiling software depends on a myriad of factors. Describing the dependencies and the build environment in their entirety is difficult, but necessary to reach an output that is always the same for every compilation.**

With the functional package manager Guix, software packages are described in the functional programming language Guile Scheme, which makes identifying dependencies simple. Guix also provides ways to describe the build environment such that it can easily reproduced. These factors combine to allow wholly reproducible builds.

We packaged a piece of moderately complex software for Guix, and have found that this is a suitable way to achieve a reproducible build.

***Index Terms*—reproducibility, guix, functional package management**

1. Introduction

It is a difficult task to build and run software reproducibly, or even reliably. This is because both compilation and execution depend on many inputs. These inputs, commonly called "dependencies", are required during build time (such as compilers and build systems) and during runtime (such as interpreters or shared libraries).

To add complexity, software also usually depends on exact versions of their dependencies. As an example, a program written to run with version 2.7 of the python interpreter may not work with version 2.6, if the program uses new features from 2.7. It may also not work with version 3.0, since python syntax between the versions is generally not compatible. Here, it is therefore not enough to state "python" as a dependency, but also the exact version.

The problem is made more difficult still because dependencies also have dependencies. To guarantee that a piece of software is built and run deterministically, it is necessary to have a system that defines software in such a way that it is possible to determine all recursive dependencies. Such a system would ideally also have a simple way of describing the entire environment, i.e. operating system, with all its software and specific versions, in which the build was performed.

Guix is a package manager and operating system that provides these features. In this paper, we set out to describe package management with Guix and how it can help make build processes reproducible in Section 2. We then package a concrete piece of moderately complex software with Guix in Section 3. In Section 4, we are then able to evaluate this packaging process in terms of complexity, and determine whether Guix is suitable for reproducibly building software. Section 5 briefly lists previous uses of Guix in academia, and a conclusion is reached in Section 6.

2. Guix

Guix is a package manager introduced in 2013 [1]. As of the time of writing, it includes 21436 packages of free software. It sets itself apart from other package managers such as `pacman` or `RPM` as a functional package manager. This difference lies in how packages are defined.

2.1. Functional Package Management

Packages, i.e. the building and installation process, are represented as pure functions, in the sense of functional programming. A pure function is a function with the following properties:

1) The function will always evaluate to the same value given the same input (i.e. it is deterministic).
2) It is free of side effects.

In the context of building software, the first property means the build processes do not depend on the state and available dependencies in the operating system,

The inputs (source code, all dependencies) for building software can be considered to be the function's parameters. The finished build is considered to be the result of evaluating the function. In Guix, this is achieved by creating a container for each build that is separated from the host OS.

2.2. Building packages

A build daemon builds packages in a `chroot` environment. It is used because it provides a lightweight way to control which programs a process uses and has access to by changing the apparent root directory [2]. Inside this container, only the dependencies explicitly listed in the package definition are visible. This ensures that the evaluation stays pure, and does not depend on the state of the host system. It is the build daemons task to configure an environment that includes not only libraries and headers needed for compilation but also explicitly defined build systems and compilers.

Now it is clear how Guix differs from other, imperative package managers. Instead of packages modifying the

 doi: 10.2313/NET-2023-11-1_02

state of the system unpredictably, as with privileged shell scripts, and depending on the packages installed in the system, installing and updating packages is instead transactional. This means that these processes can be recreated and rolled back easily. Also, because building is mediated through the daemon, and installation only requires linking to the built binaries, all actions are generally unprivileged. This is not the case for imperative managers, which need to move binaries into protected locations, such as `/bin`.

Instead of these traditional locations, all package build results are stored in a central location, the *store*. Each directory contained is prefixed by a cryptographic hash over the "function inputs". The *store* acts as a "cache of function results" [1] so that repeated evaluations of a package can be substituted by its result.

When a user wishes to install a package, they may do so by issuing an unprivileged call to the build daemon, which produces the packages inputs and dependencies, and evaluates the package function. It then stores the resulting binaries, libraries, and header files in the *store*. Links to these results are then added to the user's *profile*, a directory containing user-specific versions of the directories usually found in the root directory, such as `bin`, `etc`, or `include`.

2.3. Guile Scheme

Guix defines packages in the functional programming language Guile Scheme. While Guix's predecessor Nix shares the same mechanism for building and managing packages, Scheme is what separates them. Nix featured a different, less intuitive domain-specific language, called the "Nix expression language" [3]. The developers of Guix intended Scheme to offer simplicity so that developers could "help grow and maintain a large software distribution" [1].

These Schemes explicitly state the inputs required to build and run the software. Things that are considered inputs in that sense include other packages providing dependencies, arguments for the build system, as well as source code (from git or tarballs) [4].

Because other packages are also inputs, and they too are defined as Schemes, Guix can easily traverse and describe all dependencies of a defined package. This is particularly useful when the build daemon needs to determine which packages need to be available in the `chroot` environment during build time. It constructs all inputs, and their inputs recursively. It can then determine which of these inputs are already present in the `store`, and build all those that are not.

2.4. Channels

Collections of Guix packages are organized into git repositories called channels [5]. These channels contain collections of Scheme (`.scm`) files with package definitions. Additionally, they may also define the URL of a CI server to download pre-built packages from, patches needed for certain packages to be built, channel dependencies (other channels that are required for this one), and keys of the channel authors to authenticate commits.

While Guix includes a default channel by default, users may add additional channels. Contributors may also choose to define and publish their own channel by simply making a git repository available online.

Channels also make it possible to easily describe the state of a system with its available packages. At a particular point in time, all information that is needed to recreate a system is the URLs of its channels, as well as the commit that represents the channels' current state.

3. Packaging LLVM-LC3

The goal of this section is to describe the process and assess the complexity of packaging software for Guix. We set out to build the code generator of the LLVM toolchain with the ability to generate code for the "Little Computer 3" (LC3) educational assembly language.

The LLVM toolchain is comprised of a frontend and a backend. The **frontend** translates code from a high-level language to an intermediate representation (IR). There exist frontends for languages such as C/C++ (with `clang` as frontend), Haskell (`kaleidoscope`), or Go (`llgo`). This IR may then be translated into the target instruction set by the **backend**. Our goal is to compile this backend, and include support for LC3 as a target instruction set.

3.1. Understanding the Build Process

The source code for the LLVM backend is hosted on GitHub as part of the LLVM Project. For it to support a target instruction set, a developer needs to define the procedures on how to translate the IR into that specific machine language. Several such implementations that are already included with the backend can be found in the directory `llvm/lib/Target`, such as x86, ARM, or RISCV, with each being represented by a directory with the target's name.

The implementation of the LC3 machine, hereafter referred to as `lc3-target`, is not included, but is fortunately made available by a user on GitHub. Unfortunately, little information on how to include his implementation into the LLVM backend is provided. The trivial approach of adding the implementation into `Target/LC3` is unsuccessful. Two important pieces of information are missing:

1) Where does the backend's code need modifications for the `lc3-target` to be supported?
2) What version of the backend is this `lc3-target` intended for?

We now proceed to identify the steps to build the backend and answer these questions through examining clues and repeatedly invoking the build process after slight modifications. Answering these questions together proves difficult. When a compilation error occurs, it is not immediately clear whether it stems from an incorrect backend version, or from some necessary modification that has not yet taken place.

3.1.1. Modifications in the Backend. The backend's source code needs to be modified in three places. In two places, this is related to adding some members to enums that are used by the `lc3-target`. A modification must also be made in the `CMakeLists.txt` in the `Target` directory, for the build system to pick up the directory in which `lc3-target` resides.

 doi: 10.2313/NET-2023-11-1_02

3.1.2. Identifying the Version. We identify the version of LLVM that `lc3-target` was developed for by narrowing down the window of possible versions. Since the last commit in LC3's history is from July 24, 2016, all versions greater than 3.8 are discarded. From there, we work backward to find the correct version. Version 3.8 is discarded because `lc3-target` uses a function that was removed in that version. The next lower, version 3.7, is found to be the correct one.

3.1.3. Finding the Compiler. Even after the correct version is found, compilation still yielded an error within `lc3-target`. An implicit conversion from a `unique_ptr` to `bool` is impossible. Using version 5 of the gcc compiler instead of the latest version fixes this error, which is a known bug with old versions of llvm[1].

3.1.4. Informal Process. After successfully building and verifying the functionality of the backend, we identify the following informal steps.

1) Fetch the source code of the LLVM Project (release 3.7)
2) Fetch the source code for `lc3-target` implementation into LLVM's `Target` directory
3) Register the `lc3-target` within the backend's source code.
4) create a build directory
5) call `cmake` with certain variables to generate build files
6) build llvm

We also identify that `python 2.7` is required to generate the build files. This informal process now must be formalized into a Scheme, defining it as a package.

3.2. Defining a Scheme

Guix features a high-level Scheme data type for representing a `package`. Documentation for it and the rest of this section can be found in the Guix cookbook [5]. This data type has some self-explanatory fields that are just strings, such as the `name`, `version`, or `description`.

3.2.1. Fetching source code. Origins of source code, such as remote git repositories, tar-balls, or local files, are represented as the `origin` data type. A `package` has only one `source` field, which can be considered to be the primary `origin` of the package. In our case, however, two origins of source code are needed. Fortunately, additional `origins` can be added to a package in the `inputs` field, which contains a list of dependencies of the package.

3.2.2. Modifying source code. `origin` objects also include a `patches` field, which contains a list of patch files to be applied to the code. This is useful in our case, as it allows us to modify the code from the official llvm repository to register `lc3-target`.

1. https://github.com/digego/extempore/issues/318

3.2.3. Build systems. Guix offers many standard build systems as pre-defined `build-system` objects. They represent common ways to build software. The `build-system` field of the package contains the system to be used for the package. Depending on its value, a different sequence of commands will be executed inside the build environment. In our case, we use the `cmake-build-system`.

Sometimes modifications must be made to that idealized build process. For this purpose, it is divided into build phases, which may be edited in the Scheme. This is necessary in our case as well. In particular, the source code of `lc3-target` needs to be moved to the `Target` directory. This can be done by modifying the `configure` phase of the build system. In this phase, all the `origins` have already been fetched into the build environment. Before the phase is executed, we insert a hook that copies the `lc3-target` code to the correct location.

3.3. Adding to a Channel

Adding a package to a channel only involves adding the Scheme file to the git repository. It is also necessary to adjust the module definition at the top of the file to reference the channel.

If, like in our case, the build process involves patches from `.patch` files, they must also be added to the repository. Patches are found during building using `search-pathes` from the channels root, so the path of the patch files referenced in the Scheme must also be adjusted to be relative from there.

Finally, for Guix to recognize this channel, it is added to the user's channels, and `guix pull` needs to be executed to make packages from the channel available to build.

4. Evaluation

After the process of making a software available as a package in Guix, we are now able to discuss whether this has brought us closer to the goal of reproducibility.

4.1. Reproducibility

When trying to get the compilation to finish successfully during the first, exploratory stage (3.1.3), we encounter a roadblock that has an implication for whether the package builds deterministically. With the most recent version of gcc available in Guix at the time of writing, version 10.3.0, compilation fails due to an implicit cast. With an earlier version such as 5.5.0, the compilation succeeds.

In the Scheme defining the package, it is not required to specify the version that should be used. The following line implicitly includes the most recent version of the `commencement` package, which puts the most recent version of the gcc toolchain into the build environment.

```
#:use-module (gnu packages commencement)
```

It is therefore possible to define a package in a Scheme for Guix, that builds and installs correctly when the most recent version of gcc accepts the implicit cast. However, when a newer version of gcc treats this as an error, the build fails.

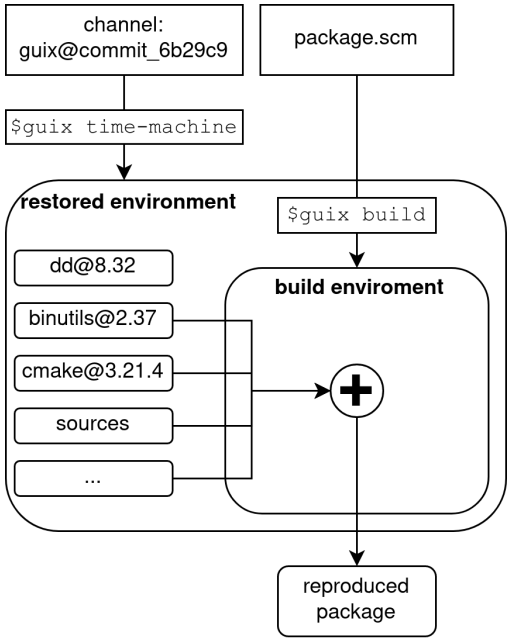


Figure 1: The same built package is reproduced from just the channel state and the package Scheme.

It is clear from this example that simply defining a package in a Scheme is not enough to make it reproducible. To actually compile with reproducibly, the packager's current Guix environment needs to be described. This includes the versions of all input packages.

For reproducing sofware, the environment must be recreated, and the package built within it. Thankfully, Guix includes a utility to do just this, called `guix time-machine` [4]. A conceptual overview of its use for this purpose is illustrated in figure 1.

To summarize, reaching the same build result each time is the result of two of Guix's features. First, with help of the Scheme package format, `guix build` is able to determine all transitive inputs of the software, down to the operating system. Second, the state of the Guix environment, including the packages at the version the software was built at, can be recreated with the `guix time-machine`. The time machine only requires the description of Guix's channels at that time for this, i.e. a git commit (`6b29c9` in this example), since channels are git repositories.

4.2. Complexity

It is a goal of the Scheme format for packages to be "purely declarative in common cases", so as to be "directly usable by packagers with little or no experience with Scheme." [1]. For such common cases, a Scheme can intuitively be constructed with help from the examples in the Guix cookbook. However, for non-obvious cases, such as ours, where a git repository needs to be fetched and copied to a specific location, the documentation does not provide an easy recipe. In cases such as these, it is very useful to browse the default channel's other package definitions. It is likely that the special case has been dealt with by another packager in the past. Guix's package still appear less complex than Nix's though, as can be seen by comparing the definitons for the same packages, like `llvm`.

To aid with debugging during the definition of the Scheme, the command `guix build` with the `--keep-failed` switch is useful. It allows building a package without a channel and also keeps the build environment to examine, which helps determine the cause of failures.

The most challenging part of our example is to determine the informal building steps and dependencies before they are translated into a Scheme. Because the developer of `lc3-target` has not provided any information on the llvm version or modifications to llvm's source code, a process of trial-and-error is necessary.

5. Related work

Guix has been used previously in scientific work. In their paper from 2015, Courtés and Wurmus explore the use of Guix in high-performance computing [6]. They show that its structure, with unprivileged users being able to build packages through calls to the user daemon, benefits environments where many users share a cluster. It is also useful because Guix packages do not depend on parts of the host system, making moving them to new clusters a less involved process. With Guix, it is also more straightforward for them to define variants of software for different use cases. In a specific example, they define different variants for a package for developers and users.

There is also use for Guix in bioinformatics. In 2018, Wurmus et al. introduce PiGx, a tool for developing pipelines that transform raw experimental data into reports [7]. PiGx is packaged using Guix, which makes it not only reproducible but also transparent. The specific dependencies and packages used can be easily described and analyzed, which would more complex if a virtual machine is used to recreate the software environment.

Recently, in 2022, work has also been done by Batten et al. as part of the CARRV workshop to use Guix for packaging RISC-V software, as well as hardware simulators like gem5 [8]. RISC-V software is mostly cross-compiled as of yet, posing the problem that toolchains for it need to be defined. This again has many complex inputs, making packaging with Guix suitable.

6. Conclusion

In this paper, we have discussed the internals and mechanisms of the Guix package manager, and what makes its approach useful for reproducibility in contrast to other package managers. Furthermore, we have shown in a concrete example the process of packaging software, and described in detail the process of learning the dependencies of largely undocumented software. We aim to make this package available in the channel `guix-past`, which is managed by France's National Institute for Research in Digital Science and Technology (INRIA). Furthermore, we have provided a recipe for using the `guix time-machine` to recreate the package environment to build a piece of software with the same inputs each time, making it reproducible.

References

[1] L. Courtès, "Functional Package Management with Guix," *CoRR*, vol. abs/1305.4584, 2013. [Online]. Available: http://arxiv.org/abs/1305.4584

[2] *chroot(2) - Linux man page*, Free Software Foundation. [Online]. Available: https://linux.die.net/man/2/chroot

[3] E. Dolstra and A. Löh, "NixOS: A Purely Functional Linux Distribution," in *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 367–378. [Online]. Available: https://doi.org/10.1145/1411204.1411255

[4] K. Hinsen, "Reproducible computations with Guix," 2020. [Online]. Available: https://guix.gnu.org/en/blog/2020/reproducible-computations-with-guix/

[5] R. Wurmus, E. Flashner, P. Neidhardt, O. Pykhalov, M. Brooks, M. Karpezo, B. Waegeneire, A. Batista, C. Lemmer-Webber, J. Branson, M. Couroyer, and L. Courté, "GNU Guix Cookbook," 2019. [Online]. Available: https://guix.gnu.org/en/cookbook/en/

[6] L. Courtès and R. Wurmus, "Reproducible and User-Controlled Software Environments in HPC with Guix," in *2nd International Workshop on Reproducibility in Parallel Computing (RepPar)*, Vienne, Austria, Aug. 2015. [Online]. Available: https://hal.inria.fr/hal-01161771

[7] R. Wurmus, B. Uyar, B. Osberg, V. Franke, A. Gosdschan, K. Wreczycka, J. Ronen, and A. Akalin, "PiGx: reproducible genomics analysis pipelines with GNU Guix," *GigaScience*, vol. 7, no. 12, 10 2018, giy123. [Online]. Available: https://doi.org/10.1093/gigascience/giy123

[8] C. Batten, P. Prins, E. Flashner, A. Isaac, J. Nieuwenhuizen, E. Zarraga, T. Ta, A. Rovinski, and E. Garrison, "The Case for Using Guix to Enable Reproducible RISC-V Software & Hardware," 2022.

 doi: 10.2313/NET-2023-11-1_02

LXC Container Between `cgroups` v1 and v2: a Performance Evaluation

Alexander Daichendt, Florian Wiedner*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: daichend@net.in.tum.de, wiedner@net.in.tum.de, andre@net.in.tum.de

***Abstract*—The cgroups feature of the Linux kernel is widely used by lightweight virtualization technologies such as Docker or LXC to provide resource isolation. Recently, cgroups underwent a significant revamp from version 1 (v1) to version 2 (v2). Researching the performance difference between these two versions in terms of network latencies enables the usage of containers in time-critical applications. Previous work ignored cgroups as a potential source of latency in packet-processing systems. In this paper, we measure the performance difference between cgroups v1 and v2 in isolation using commodity hard- and software. Our experiments show that the two versions achieve the same degree of isolation, but the tail latencies of v1 are higher, which can be explained by a more efficient, 2.4 % less instruction-consuming implementation of v2. Therefore, we recommend the use of v2 for low-latency lightweight virtualization network deployments wherever possible.**

***Index Terms*—low latency, container, lxc, virtualization, dpdk, cgroups, packet processing**

1. Introduction

From inter-vehicle communication in self-driving cars to the coordination of assembly lines, critical applications require technology to operate at peak performance. In such scenarios, even the slightest delay can result in catastrophic consequences. That is why network latencies are a crucial factor in enabling the interaction and coordination of sensitive applications. To achieve the lowest and most stable network latencies, it is crucial to invest in high-quality networking equipment and thoroughly review the entire software stack.

A common way of handling increasing complexity is to compartmentalize different software components into smaller pieces and run them in isolated environments. This can be achieved through virtualization, using either heavyweight virtual machines or lightweight containers. A deeper understanding of the inner workings of the chosen virtualization technique is necessary to optimize for low-latency networking. Two essential features for enabling such optimizations are namespaces and control groups (cgroups).

Namespaces allow processes to have isolated and independent views of the system resources, such as the network, filesystem, or process IDs. cgroups provide a way to limit, allocate, and prioritize system resources among processes or groups of processes. Initially, cgroups were released in 2007 in kernel 2.6.24 [1] as version 1 (v1). They were completely revamped [2] with a second version (v2) released in kernel 4.5. Since cgroups v1 and v2 differ in their implemented features, this paper highlights the differences between them. Furthermore, we show the impact of the cgroup version on network latencies with experimental measurements.

The paper is structured as follows: Section 2 presents related work. In Section 3, we provide background information on cgroups and Linux containers (LXC). Section 4 details the specific optimizations we apply to increase isolation. Section 5 discusses the experimental setup and measurements. Finally, we summarize our findings in Section 6 and propose future work.

2. Related Work

In our previous work [3], we expanded the capabilities of HVNet by Wiedner et al. [4], a framework for orchestrating low-latency experiments on a single host with KVM, by incorporating LXC containers, initially developed by Wiedner et al. HVNet automates the setup of all involved hosts, configures them for low-latency networking, and synchronizes the measurement scripts. Virtual networking topologies can be defined in text files. Our measurements [3] of network latencies for LXC containers with cgroups v2 demonstrated comparable performance to VMs but with occasional spikes in tail latencies. To address this issue, we found that using a real-time kernel is effective. Our prior implementation and research serve as the foundation for this paper, where we introduce a new feature in HVNet to switch between cgroups v1 and v2 and compare the cgroup versions to evaluate their performance.

Abeni et al. [5] propose a real-time scheduler for the Linux kernel that is aware of cgroups, making it compatible with Docker and LXC. They demonstrate experimentally that their scheduler delivers lower average response times for a task set than KVM but with similar worst-case latencies. Notably, their scheduler is capable of migrating processes to another core that has processing time left; a feature unavailable in KVM-based systems since the hypervisor has no access to the scheduling of a VM guest. However, it provides lower network latencies for cgroup-based virtualization. Although their work shows promise, it may not be directly applicable to our setup. In our setup, a single core processes all packets of a network interface card (NIC) in userspace with the Data Plane Development Kit (DPDK). Process migration between different cores is not anticipated.

 doi: 10.2313/NET-2023-11-1_03

Listing 1: cgroups v1 hierarchy [6]

```
/sys/fs/cgroup
├── cpuset
│   ├── cgroup0
│   │   ├── cpuset.cpus
│   │   └── cpuset.mems
│   └── cgroup1
│       └── cpuset.cpus
└── memory
    └── cgroup0
        └── memory.limit_in_bytes
```

Listing 2: cgroups v2 hierarchy [2]

```
/sys/fs/cgroup
├── cgroup0
│   ├── cpuset.cpus
│   ├── cpuset.mems
│   └── memory.max
└── cgroup1
    └── cpuset.cpus
```

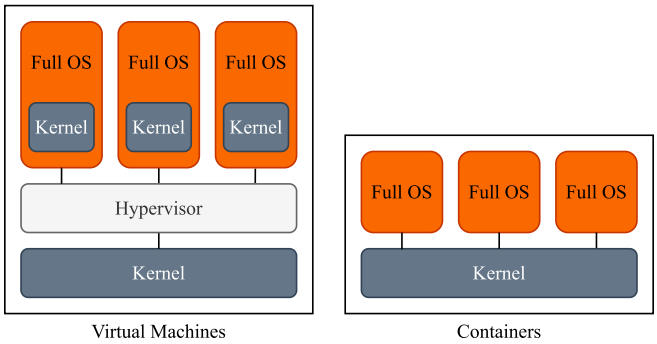


Figure 1: Architecture of VMs and containers [3]

3. Background

Section 3.1 presents a short introduction to cgroups, highlighting their key features and differences between v1 and v2. Section 3.2 introduces the concept of containers and Linux Containers (LXC), the container implementation we rely upon.

3.1. cgroups

cgroup is a Linux kernel feature that assigns resources such as CPU time, memory, and I/O between processes. Resources can be limited, prioritized, and isolated, enabling administrators fine-grained control over the system. There are scenarios where it is desirable to guarantee that one critical process has access to resources. For example, a background cronjob should not compete for resources with a web server and potentially negatively affect the latency of a request. With cgroups, the administrator can prevent resource contention by guaranteeing resources to the webserver and limiting non-critical processes.

Both cgroup versions are mounted in the same location, `/sys/fs/cgroup`, but differ in their hierarchical structure. Listing 1 shows the hierarchy for v1, where each controller is represented by a separate mount point, and a cgroup must be created for each controller individually. In contrast, Listing 2 depicts the same hierarchy for v2. A unified, hierarchical structure represented by a single mount point of type `cgroup2` holds all controllers and groups. Each group can hold any number of enabled controllers.

In v2, it is no longer possible to assign a process to an internal node of the tree hierarchy as claimed by Down [7]. These properties can be verified on any modern Linux-based system by inspecting the output of `systemctl status`. The "no inner process" node clears up the hierarchy and makes it easier to understand.

In addition, several inconsistencies have been addressed in cgroups v2, leading to a higher degree of standardization. For instance, the renaming of `memory.limit_in_bytes` to `memory.max` is evident in Listing 1 and 2. These standardizations have been applied to all thresholds, resulting in a more uniform and consistent naming scheme.

Important for our network latency performance analysis is the scheduler load-balancing option. Scheduler load balancing is a feature where a process may be migrated to a different core to balance the load equally in a multicore system [8]. In latency-critical applications, a single context switch can cause a spike in latency. In v1, this behavior can be disabled by modifying the file `cpuset.sched_load_balance`. However, in v2, this option was initially removed. Only recent kernels ($\geq$ 6.1) support this option which was introduced by Waiman [9]. This paper disables scheduler load balancing for v1 and compares it to cgroups v2 with load balancing. Testing this option for cgroups v2 is out of scope for this paper due to missing infrastructure for testing the latest kernels.

Finally, another change is that in v1, each thread of a process could be assigned to a different cgroup [7]. This behavior is considered confusing and unnecessary and is no longer present in v2.

3.2. LXC Containers

Containers are a lightweight alternative for virtualization. They are considered to be operating system level because they share the host kernel and operate on the same level as any other process in userspace. Figure 1 highlights this architectural difference between VMs and containers. A container does not virtualize its own kernel, while a VM does. Furthermore, no hypervisor is required. Sharing the kernel with the host and other containers has implications for isolation. However, modern kernels offer features that help to build isolated systems. The most important features are cgroups and namespaces.

LXC is a low-level container runtime being in active development since 2008. It provides a minimalistic feature set to remain lightweight with minimal overhead. Userspace tools for managing LXC containers are available. A C or Python API is available for controlling LXC for more advanced use cases. One downside of LXC is that convenience features such as layered images or orchestration are missing entirely. LXC images are typically larger than Docker images since they snapshot the entire root

filesystem of an OS installation - including the installed libraries.

We use LXC 4.0, the userspace tools, and the Python API for our implementation. To evaluate the performance difference of cgroups v1 and v2, we extend an existing framework for low-latency measurements: HVNet [4].

4. Implementation

Our original plan was to implement cgroups v1 and v2 on Debian Buster to enable a more seamless comparison with previous work by Wiedner et al. [4] and Gallenmüller et al. [10], [11]. However, Debian Buster runs on kernel 4.19, which does not yet include the cpuset controller [12]. Without the cpuset controller, the container is unisolatable from the rest of the system, making a performance comparison between cgroups v1 and v2 meaningless. Therefore, we focus our efforts on Debian Bullseye, which supports the legacy cgroups v1 and a more mature implementation of cgroups v2.

To switch between the two cgroups versions, a startup flag `--lxc-enable-cgroup-v1` is implemented in HVNet [4]. By setting this flag, the container host is booted with the kernel parameter `systemd.unified_cgroup_hierarchy=0`. This parameter disables the unified cgroups v2 hierarchy and enables the legacy cgroups v1.

The process isolation methodology differs between cgroups v1 and v2, but both achieve the same goal. For v2, we utilized two methods: first, the feature `cpuset.cpus.partition`, which removes CPU cores of a child from the parent cgroup. Second, we use systemd to restrict all processes to CPU cores unused by the container through the command `systemctl set-property user.slice AllowedCPUs=0-23;27-31`. In contrast, neither of these features are available with v1. Instead, we followed the instructions suggested by Weisbecker [13]. First, we create a new housekeeping cgroup and restrict its access to CPU cores, ensuring no cores are shared between housekeeping and the container. Next, we move all processes, including kernel threads unrelated to the container into this cgroup using the Python tool `cset` [14]. It is worth noting that the init process with PID must remain in the root cgroup; otherwise, it would be impossible to start a container. By applying these optimization techniques, we have ensured that the processing cores are fully isolated.

5. Evaluation

In the first Section 5.1, we introduce the experiment setup in detail. Subsequently, in Section 5.2 we present the findings from our measurements and provide a comprehensive analysis and discussion of the results.

5.1. Experiment Setup

Our experimental setup follows both HVNet [4] and our previous work [3]. Figure 2 provides an overview of the configuration for the three hosts involved in the experiment: the Device under Test (DuT), the Timestamper, and the load generator (LoadGen). To generate packets on the LoadGen, we utilize MoonGen by Emmerich et al. [15], a flexible high-performance packet generator written in Lua. The Timestamper is connected to the ingress and egress lines via passive optical terminal access points (taps), which add a negligible, constant delay. The DuT runs a single LXC container with direct access to the ingress and egress interfaces and runs a minimal DPDK L2 forwarding application.

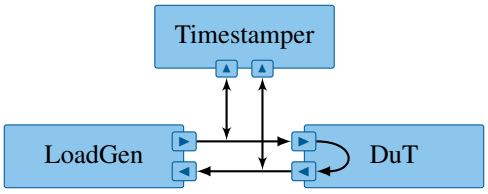


Figure 2: Experiment setup [3]

The LoadGen features an Intel Xeon Silver 4116, 192 GB RAM, and a dual-port Intel 82599ES 10-Gigabit SFP+ NIC connected to the DuT with optical fibers. The DuT is equipped with an AMD EPYC 7551P, 128 GB RAM, and a dual-port Intel X710 10Gbe SFP+ NIC. The Timestamper is outfitted with an AMD EPYC 7542, 512 GB RAM, and a dual-port Intel E810-XXVDA4 25-Gigabit flashed to 10-Gigabit NIC, providing 1.25 ns precision.

To automate and make the measurements reproducible, we use the plain orchestration service (pos) by Gallenmüller et al. [16]. This service enables us to control boot parameters, power status, and images of bare-metal hosts and VMs hosted by libvirt, utilizing IPMI. In our previous work [3], we developed virtualLXCBMC [17] to integrate LXC with pos, which enables us to control LXC containers with IPMI. However, since a container does not have its own kernel, it is impossible to set boot parameters.

Each packet carries a unique identifier to precisely evaluate its network latency. The packets are timestamped by their respective NICs. The Timestamper matches a packet on the ingress and egress and measures the duration. This methodology enables us to measure the processing latency without introducing latency by the measurement process itself. Subsequently, the Timestamper generates pcap files which scripts process further.

In our measurements, we utilize minimal-sized packets of 64 B as the processing cost of a single packet remains constant, irrespective of its size [15]. Therefore, the number of packets, not their size, is the predominant factor contributing to processing delays. We measure with a packet rate of 1.52 Mpkt/s corresponding to 825 Mbit s^{-1}, and 6.24 Mpkt/s corresponding to 3.39 Gbit s^{-1}, like in our previous work [3]. On the DuT we use Debian Bullseye with a real-time kernel 5.10.

5.2. Results

To measure the tail latencies of cgroups v1 and v2, we use the setup of Section 5.1, the optimizations described in Section 4 and [4]. Each experiment is repeated three times; the worst case is reported in this paper. We have made instructions for reproduction and additional measurement data available for inspection[1].

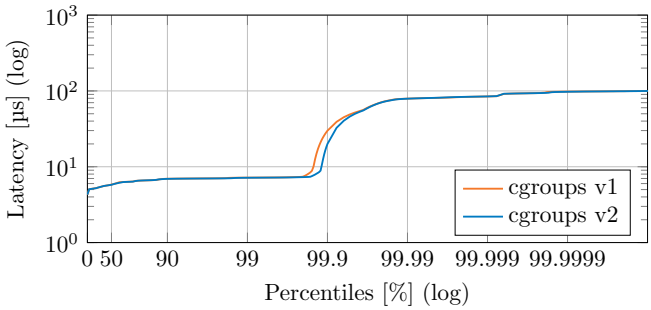


Figure 3: HDR histogram of cgroups v1 and v2 with 1.52 Mpkt/s on Debian Bullseye with real-time kernel.

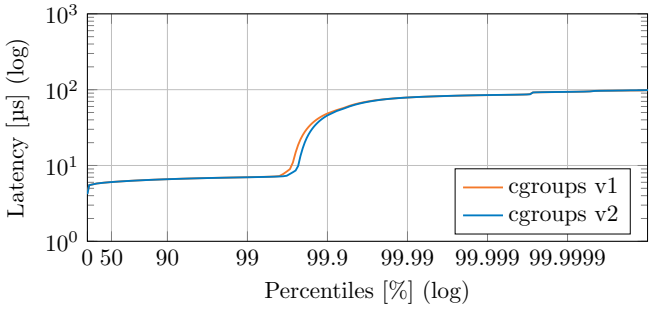


Figure 4: HDR histogram of cgroups v1 and v2 with 6.24 Mpkt/s on Debian Bullseye with real-time kernel.

The latency differences between cgroups v1 and v2 on Debian Bullseye with a real-time kernel, with a packet rate of 1.52 Mpkt/s, are shown in Figure 3. Both versions exhibit a nearly identical latency trend, with a slight difference emerging between the 99^{th} and 99.9^{th} percentiles. Specifically, the latency with v1 at the 99.9^{th} percentile is slightly higher than with v2. However, towards the 99.99^{th} percentile, the network latencies of both versions match again.

Figure 4 presents the result of the same experiment with a higher packet rate of 6.24 Mpkt/s. The same trend as in Figure 3 is visible, albeit the spike in tail latency occurs slightly earlier. This behavior is expected, as the packet rate is four times higher than before. Likewise, v1 exhibits higher worst-case network latencies.

The 5000 worst-case latencies, as shown in Figure 5, are similarly distributed for cgroups v1 and v2, as the HDR histograms already suggested. There are slightly more outliers for v1, indicating that more packets are affected by higher processing delays than for v2. Our remaining measurement data suggests that extreme outliers, like those seen for v2 at the 34^{th} second of measurement time, are more prevalent for v2.

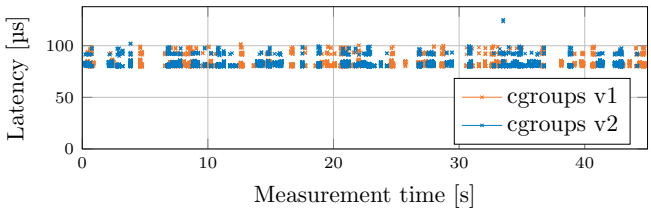


Figure 5: 5000 worst-case latencies of cgroups v1 and v2 with 1.52 Mpkt/s on Debian Bullseye with real-time kernel

TABLE 1: Performance analysis with `perf stat -B` between cgroups v1 and v2.

cgroups	instructions	branches	migrations
v1	674×10^9	98×10^9	297
v2	659×10^9	96×10^9	148

We verify the claim that cgroups v2 has a more efficient implementation by measuring the number of instructions executed with the Linux performance analysis tool `perf` [18]. The measurement includes the container startup, 60 s packet forwarding at 1.52 Mpkt/s, and the shutdown. The experiment is repeated three times, taking the average value. The resulting data is presented in Table 1 and in the reproduction collection[1]. We observe that for cgroups v1, the number of instructions executed is about 2.4 % higher, and about 2.2 % more conditional branches are executed compared to v2. The difference in process migrations of 148 in v2 and 297 in v1 is noteworthy. Given that we disabled scheduler load balancing, this finding is unexpected.

While these differences seem small, it is crucial to note that our recorded difference in latencies only occurs at the 99.9^{th} percentile, which concerns only a fraction of all packets.

6. Conclusion and Future Work

The Linux kernel is a constantly evolving system, with bug fixing and continuing feature expansion. However, with these rapid changes, there is no extensive research available to evaluate the changes. To ensure the reliability and safety of using containers and cgroups in low-latency systems like self-driving cars or airplanes, detailed studies are necessary. In this paper, we have demonstrated that cgroups v2 is a superior choice for low-latency networking. While the latency behavior is identical to cgroups v1 up to the 99^{th} percentile, v1 performs worse with more packets having higher latency. Additionally, v1 consumes 2.4 % more instructions for the same workload, indicating its less efficient implementation. Therefore, we conclude that cgroups v2 is the better choice for low-latency systems that require high performance and reliability. It is worth noting, however, that v2 lacks some of the features of v1, and upgrading to a more modern kernel may not always be possible without additional costs. Systems in production often utilize operating systems with long release cycles, which makes changes to packaged software like the kernel expensive.

As part of our future work, we aim to enhance and automate measuring and analyzing the instructions consumed by a container, the underlying technology, and the applications inside. Automation would enable us to assess the performance of different enabling technologies like cgroups more effectively. Additionally, we are interested in testing the kernel 6.1, which incorporates the new cpuset partition type `isolated`, and comparing it against cgroups v1 `cpuset.sched_load_balance`.

1. https://wiedner.pages.gitlab.lrz.de/iitm-seminar-daichendt-reproducibility/

 doi: 10.2313/NET-2023-11-1_03

References

[1] J. Corbet, "Notes from a container," *LWN.net*, 10 2007, accessed on March 23, 2023. [Online]. Available: https://lwn.net/Articles/256389/

[2] R. Rosen, "Understanding the new control groups API," *LWN.net*, 3 2016, accessed on March 23, 2023. [Online]. Available: https://lwn.net/Articles/679786/

[3] A. Daichendt, "Lightweight low-latency virtual networking," August 2022, B.Sc. thesis, found at https://gitlab.lrz.de/wiedner/iitm-seminar-daichendt-reproducibility/-/blob/master/data/thesis.pdf.

[4] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted virtual networking on a single physical host," in *IEEE INFOCOM WKSHPS: Computer and Networking Experimental Research using Testbeds (CNERT 2022) (INFOCOM WKSHPS CNERT 2022)*, Virtual Event, May 2022.

[5] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," *ACM SIGBED Review*, vol. 16, no. 3, pp. 33–38, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3373400.3373405

[6] M. Kerrisk, "cgroups(7) - linux manual page," https://www.man7.org/linux/man-pages/man7/cgroups.7.html, 2021, accessed on April 1, 2023.

[7] C. Down, "cgroupv2: Linux's new unified control group system," QCON London, 2017.

[8] Debian. (2020, November) cpuset(7) - linux manual page. [Online]. Available: https://manpages.debian.org/bullseye/manpages/cpuset.7.en.html

[9] L. Waiman, "cgroup/cpuset: Add a new isolated cpus.partition type," https://github.com/torvalds/linux/commit/f28e22441f353aa2c954a1b1e29144f8841f1e8a, Sep. 2022.

[10] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems," in *3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking (HiPNet 2021)*, Izmir, Turkey, Oct. 2021.

[11] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G URLLC: A case study on low-latency intrusion prevention," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 35–41, Oct. 2020.

[12] H. Tejun, "Control Group v2," https://www.kernel.org/doc/html/v4.19/admin-guide/cgroup-v2.html, Oct. 2015, accessed on March 23, 2023.

[13] F. Weisbecker, "CPU isolation practical example, part 5," January 2022, accessed on March 23, 2023. [Online]. Available: https://www.suse.com/c/cpu-isolation-practical-example-part-5/

[14] A. Tsariounov, "cpuset," https://github.com/lpechacek/cpuset, 2018, accessed on 2023-03-23.

[15] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference (IMC) 2015, IRTF Applied Networking Research Prize 2017*, Tokyo, Japan, Oct. 2015.

[16] S. Gallenmüller, D. Scholz, H. Stubbe, E. Hauser, and G. Carle, "Reproducible by Design: Network Experiments with pos," in *KuVS Fachgespräch - Würzburg Workshop on Modeling, Analysis and Simulation of Next-Generation Communication Networks 2022 (WueWoWas'22)*, Würzburg, Germany, Jul. 2022.

[17] A. Daichendt, "VirtualLXCBMC," August 2022. [Online]. Available: https://github.com/AnonymContainer/virtuallxcbmc

[18] "perf-stat(1) - linux manual page," https://www.man7.org/linux/man-pages/man1/perf-stat.1.html, accessed on March 23, 2023.

Common Workflow Language Execution on the I8-Testbed

Thomas Dietrich, Sebastian Gallenmüller*, Manuel Simon*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: thomas.dietrich@tum.de, gallenmu@net.in.tum.de, simonm@net.in.tum.de

***Abstract*—The Common Workflow Language (CWL) is an open standard to describe workflows in a portable way. Data analysis using workflows has increased significantly in science. Nevertheless, there has been no coordinated way to express them, resulting in multiple diverse workflow systems that are complicated to exchange. Exchange and verification are fundamental principles in research. Therefore, it is essential to share the workflows behind the findings in a standardised way so that the results are reproducible and can be confirmed. This article describes the necessary steps to install and execute CWL workflows on a testbed that uses the pos framework to manage nodes. For the execution of CWL, we chose an implementation called StreamFlow because it provides SSH Connectors. A CWL execution on this testbed utilises a combination of StreamFlow, the pos framework and bash scripts. This automation framework can execute new CWL workflows or existing ones from other researchers without a high expense. It enables data analysis with CWL and its verification. An example implementation is available at https://gitlab.lrz.de/netintum/teaching/iitm/repos/2023ss-bs/u838/-/tree/main/resources/experiment.**

***Index Terms*—experiment workflows, common workflow language, streamflow, testbed**

1. Introduction

Scientific experiments are capable of analysing data to acquire knowledge. These data analyses can consist of several steps following a fixed sequence to generate insights and meaningful output data from the input material. Computers carry out these workflows when dealing with large amounts of data or complex tasks. For this, scientists need applications that execute these workflows based on input data. Competing solutions exist from various companies that perform workflows in different ways. Baker and van Hemert's research [1] concluded that workflows become the dominant technology in describing scientific processes. As a result of this growth, there are over 300 different computational data analysis workflow systems compiled by Amstutz et al. [2].
Exchange with other researchers, mutual control, peer review and transparency are the fundamentals of scientific research. For this, workflows used in scientific methods must be traceable. However, this is complex, with multiple different systems. Repeating experiments with other systems requires a high expense and is impossible if two systems are incompatible. A new standard called the Common Workflow Language (CWL) was developed by Amstutz et al. [3] to ensure that academic principles can also be maintained when using workflows. A multi-faceted project created by Crusoe et al. [4] around the mere language strives for portably exchangeable workflows between systems with various environments, thus enabling scientists to reproduce data analyses for more transparency and better control over results. Due to these reasons, Leipzig [5] identified the CWL standard as a rising trend in his review of bioinformatic pipeline frameworks.
In addition, the CWL project provides tools to facilitate editing and viewing workflows, converting existing languages into CWL through converters and enabling execution through frameworks. The creators of CWL also provide a reference implementation called cwltool that supports local execution on Linux, Mac and Windows. Besides the reference solution, production-ready implementations with other features exist, such as the StreamFlow implementation by Colonnelli et al. [6] with various connectors. These connectors allow StreamFlow to connect to hybrid workflows using cloud computing and high-performance computers or multi-container environments like Docker containers by Merkel [7].
This paper describes the necessary steps to install and execute the CWL. It uses an implementation of StreamFlow with SSH Connectors to combine CWL workflow steps to nodes on the testbed of the Chair of Network Architectures and Services from the Technical University of Munich (I8), which Haden et al. introduced in [8]. In addition, bash scripts automate the execution of StreamFlow projects on the testbed of the I8 using the pos framework from Gallenmüller et al. [9].
The content of this paper is structured as follows. Sec. 2 briefly describes how to write a workflow in CWL, followed by Sec. 3, explaining how StreamFlow is structured and how to use it to execute the CWL. Sec. 4 uses this information to combine CWL and StreamFlow with bash scripts and the pos framework to create an experiment runnable on the testbed. Sec. 5 concludes the paper and provides an outlook on further contents of the CWL project.

2. Common Workflow Language

CWL is an open and free standard. It describes workflows in a human-readable way. CWL focuses on its community and aims to be interoperable, vendor-neutral and portable across different platforms. Reusable workflows and reproducible results can improve transparency in research. There are tools, libraries and editor plugins that already support CWL. Furthermore, you can parallelize

 doi: 10.2313/NET-2023-11-1_04

workflows with the scatter feature [4] to make them scalable. However, this paper will only deal with the basic functionalities of CWL because additions like tools or the scatter feature would exceed its scope.
CWL workflow files are written in YAML and have the ending `.cwl`. There are two essential class types: the `CommandLineTool` and the `Workflow` class. The `CommandLineTool` class is a wrapper for software tools executed with the command line. It stores the command, describes input and output and assigns them an id defined in the `Workflow` class [3].

```
1  cwlVersion: v1.2
2  class: CommandLineTool
3  baseCommand: echo
4  stdout: output.txt
5  inputs:
6    step_in:
7      type: string
8      inputBinding:
9        position: 1
10 outputs:
11   step_out:
12     type: stdout
```

Listing 1: echo.cwl

The functionality of the file above is to repeat an input message captured in a text file called `output.txt`. After specifying the `cwlVersion` and the `class` type, a `baseCommand` has to be declared. The expression in Line 4 of Listing 1 captures the messages from `stdout` in a file. The class assigns input and output names and types. The `inputBinding` describes the position of arguments after the command [3].

```
1  cwlVersion: v1.2
2  class: Workflow
3  inputs:
4    message: string
5  outputs:
6    out:
7      type: File
8      outputSource:
           echo/step_out
9  steps:
10   echo:
11     run: echo.cwl
12     in:
13       step_in: message
14     out: [step_out]
```

Listing 2: workflow.cwl

To define the workflow, the `Workflow` class uses the `CommandLineTool` classes. It specifies the input and output of the whole workflow and assigns intermediate products to steps. Inputs of workflows are typically stored in JSON or YAML files with the same name as the workflow file followed by a `"-job"` suffix. CWL is limited by the input given once the workflow has been started. Since it is not possible to add input later, care must be taken to add all the required data before starting the workflow to avoid later problems. The `workflow.cwl` file uses the `CommandLineTool` from Listing 1 to build an example workflow with one step. A complex workflow can be created, in a human-readable manner, by adding other work steps. CWL also supports additional requirements like using JavaScript with the `InlineJavaScriptRequirement` for added functionality. Installing the required packages for all extra functionalities in the setup is necessary. To evaluate, e. g. inline JavaScript expressions, `Node.js` has to be installed in addition to the basic packages, as shown in Section 4.2 [3].

3. StreamFlow

StreamFlow is a workflow manager capable of executing CWL workflows on different architectures. As shown in Figure 1, this workflow manager executes a StreamFlow file. This file is written in YAML and conventionally is called `streamflow.yml`. It consists of two parts. The first one is called "Model description files" and contains a YAML description of infrastructures that should execute the workflow steps, e. g., different servers or different nodes of a testbed.

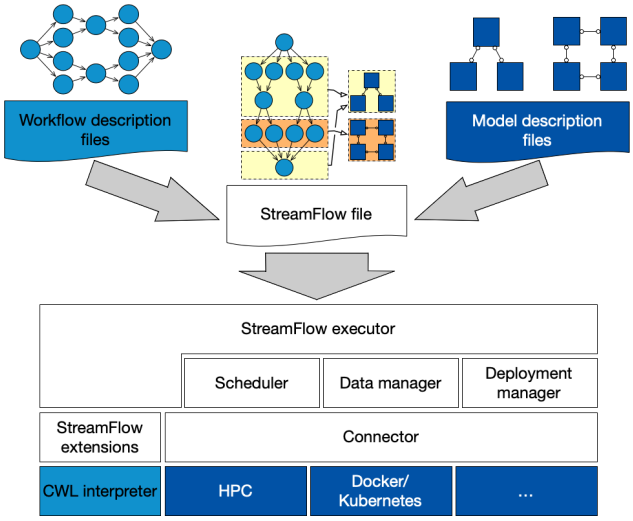


Figure 1: StreamFlow Model from [6]

The deployment diagrams in the upper right corner show that there can be multiple models with different structures within a single model description file. A model has a `type` section containing the chosen connector and a `config` field with connector-specific configuration parameters. The second part, called "Workflow description files", references a workflow file like `workflow.cwl` in Listing 2, or, as indicated by the activity diagram in the upper left corner of Figure 1, can be a complex workflow that runs in parallel steps. Input files mentioned in Section 2 and a `bindings` section are also part of these description files. The binding section describes which workflow step is deployed on which model. This section can, e. g. specify that the echo step in Listing 2, Line 10 should be executed on a specific node from the testbed mentioned in the introduction. This binding is visualised by the arrows in the illustration in the middle of Figure 1 that combine the deployment diagram and the activity diagram. As a whole, combined by the `bindings` section, these two description files result in a StreamFlow file. With this file, the StreamFlow executor can interpret CWL, schedule

tasks and deploy the workflow to connected environments while managing the data by itself [6].

This paper uses an `SSHConnector` to connect to different nodes. The names of the used nodes listed in an array called `nodes`, the username in a field `username`, and a path to the SSH key file on the nodes in `sshKey` are configuration parameters for this connector. As depicted in Figure 1, there also exist other connectors for high-performance computing (HPC) and container technologies. As mentioned in the introduction, there is, e. g. a `DockerConnector` that requires a docker image as his `config` or a `SlurmConnector` that is capable of connecting to an HPC facility orchestrated by the Slurm queue manager from Yoo et al. [10]. This `SlurmConnector` requires a hostname of an HPC facility and a username for the SSH connection to this facility. While container connectors would also be feasible on the I8-testbed, HPC connectors require an HPC facility [6].

4. Implementation on the Testbed

Once the workflow is described by CWL and the execution is managed with StreamFlow, a Bash script called `experiment.sh` is executed on the management node. It automatically orchestrates nodes from the testbed [8] using the pos framework, simultaneously installs the needed packages, prepares SSH Connections for the connectors mentioned at the end of Section 3, copies required files to the nodes and executes the StreamFlow file.

4.1. Orchestration of Nodes

The testbed consists of nodes orchestrated by a management node through the pos framework [9]. The framework can reserve selected nodes by creating calendar entries for them. It allocates the nodes for the experiment and boots them with specified images. Furthermore, pos can copy files from the management node to other nodes and remotely execute commands or executables on selected nodes. For the execution of StreamFlow, a master node is set up that uses a selectable amount of worker nodes, as shown in Figure 2. A Debian 11 (bullseye) image, e. g. is needed, since StreamFlow requires `python >= 3.8` [6].

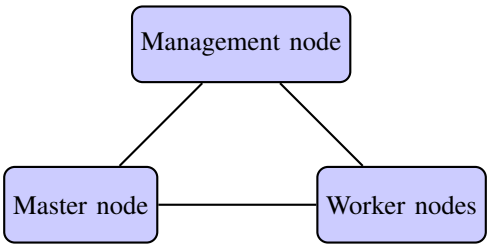


Figure 2: Testbed typology

4.2. Basic Nodes Setup

The `experiment.sh` script simultaneously executes a setup script called `setup.sh` on all selected nodes using the pos launch feature. The script displayed in Listing 3 installs the needed packages for CWL and the StreamFlow execution [9].

,

```
1 #!/bin/bash
2 apt-get update
3 apt-get install -y python3-pip
4 pip install streamflow
5 pip install --force-reinstall
      "cwltool==3.1.20220802125926"
6 curl -fsSL
      https://deb.nodesource.com/setup_19.x
      | bash - &&\
7 apt-get install -y nodejs
8
9 #add needed packages here
```

Listing 3: The setup.sh script

After updating, the script installs the Python package installer `pip` because it is used to install StreamFlow. Once `pip` is available, it installs StreamFlow. Installing StreamFlow downloads a cwltool version missing a file [11]. Therefore, it is necessary to reinstall the latest version, `cwltool==3.1.20220802125926`, without this bug. Users can add other packages required for the experiment to this list. Inline Javascript expressions, e. g. , can be used in the CWL. To evaluate these expressions, NodeSource, a distribution of `Node.js` for Debian [12], must be installed. Once all packages are installed, the master and worker nodes are processed differently. Only the master node receives the `streamflow` folder in the `experiment` directory displayed in Figure 3.

4.3. Preparation of SSH Connections

From now on, it would be possible to execute CWL files on nodes locally with the `cwltool` mentioned in Section 4.2. So it would also be possible to build further preparation steps as a workflow. However, since this would generate a lot of overhead and exceed the concepts presented in section 2, bash scripts also perform further preparations. By default, the management node can communicate with other nodes. To achieve connectivity - as shown in Figure 2 - the master node should also be able to connect to the worker nodes. Therefore, the experiment script prepares the master and worker nodes for SSH connections. The `experiment` directory contains a pre-generated SSH key pair with a private key called `worker` and a public key called `worker.pub`, as shown in Figure 3. The framework script `experiment.sh` executed on the management node remotely copies the public SSH key to all worker nodes with the pos copy feature. Then a script called `keyexchange.sh` is remotely executed on all workers with the pos remote command execution feature from the management node. This setup script appends the copied public key to the `authorized_keys` in the `.ssh` folders to approve the new SSH connection. The `streamflow` folder copied to the master node, as described

in Section 4.2, contains the private key `worker`. To copy the private key, `pos` needs read permission on this file. Therefore the private key has reading permissions for everyone (`644`). Since this would be a security gap, the main script repeals the read permissions by executing the `safeKey.sh` script remotely on the master node. As a result, the private key has the default permissions `600`. Besides the private key, the master node needs all worker nodes mentioned in his `known_hosts`. The framework script uses the command `ssh-keyscan` to gather all public keys from the worker nodes in a manually created `known_hosts file`. This file is remotely stored on the master node by the management node and replaces the existing `known_hosts` file to verify all worker nodes to the master. After this preparation, the master node can connect to all worker nodes through an SSH connection to delegate work.

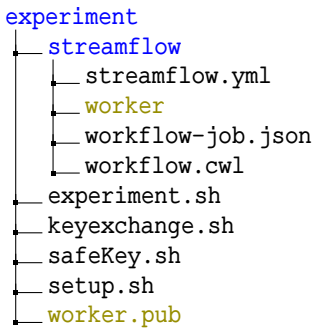


Figure 3: The experiment directory

4.4. Execution of the Experiment

The directory structure of a StreamFlow experiment could be set up as depicted in Figure 3. The `experiment` directory contains a `streamflow` subfolder that is copied to the master node from the management node. It contains the workflow files from Section 2, the StreamFlow file from Section 3 and the private SSH key. Section 4.3 explained this key with the public SSH key, the key-scripts `safeKey.sh` and `keyexchange.sh`. The main script `experiment.sh` explained in Section 4 and the `setup.sh` script shown in Listing 3 complete the experiment.
An example implementation can be downloaded from a GitLab Repository [13]. To execute the workflow, the `experiment` directory in Figure 3 can be copied to the management node of the testbed using the secure copy command `scp`. The main bash script that starts the whole experiment, "`bash experiment/experiment.sh`", is executed from the parent directory of `experiment` as the working directory. It takes a master node as the first argument and at least one worker node as the following arguments. For the workflow to function, the workers listed in the `nodes` array mentioned in Section 3 have to be passed as parameters after the freely selectable master node. When using this framework with larger projects, the hardcoded waiting time for the preparation to finish in `experimant.sh` has to be adjusted. Through the freely adjustable number of worker nodes, the script can scale with larger projects until all nodes from the testbed work to capacity if the workflows are built in a parallelisable way, e. g. , by using the scatter feature mentioned in Section 2. Additional tools required for workflow steps, can be added to the `setup.sh` script, as shown in Section 4.2. Consequently, this proof of concept can be used for further workflow experiments.

5. Conclusion

Instead of using proprietary workflow systems, complex data analysis flows can be described in the CWL standard. The StreamFlow manager can execute workflow descriptions in CWL. This manager interprets CWL and uses SSHConnectors to connect the master nodes with workers. In addition, it independently schedules deployment to workers and accomplishes data management. To automatically run the experiment on the I8-Testbed, bash scripts use features from the pos framework to orchestrate nodes, set them up, prepare SSH connections and execute the StreamFlow file. This way, data analysis workflows are portable and can be exchanged with other researchers to reuse the workflows and reproduce results. The exchangeable and verifiable CWL standard improves two necessary principles of modern research using automated data analysis and enhancing the transparency of scientific findings.
This paper is a proof of concept for a testbed framework of the CWL standard and does not cover the whole standard. There are further undiscussed CWL topics regarding CWL because it would have been too much for the scope of this paper. CWL offers, e. g. , a scatter feature to parallelise workflows to improve their scalability.

References

[1] A. Barker and J. van Hemert, "Scientific workflow: A survey and research directions," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 746–753.

[2] P. Amstutz, M. Mikheev, M. R. Crusoe, N. Tijanić, and S. Lampa, "Existing workflow systems," 2022, updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://s.apache.org/existing-workflow-systems

[3] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0," 7 2016. [Online]. Available: https://figshare.com/articles/dataset/Common_Workflow_Language_draft_3/3115156

[4] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanić, H. Ménager, S. Soiland-Reyes, B. Gavrilović, C. Goble, and T. C. Community, "Methods included: Standardizing computational reuse and portability with the common workflow language," *Commun. ACM*, vol. 65, no. 6, p. 54–63, may 2022.

[5] J. Leipzig, "A review of bioinformatic pipeline frameworks," *Briefings in bioinformatics*, vol. 18, no. 3, pp. 530–536, 2017.

[6] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "Streamflow: Cross-breeding cloud with hpc," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021.

[7] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[8] M. Haden, "I8-testbed: Introduction," in *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Summer Semester 2020*, ser. Network Architectures and Services (NET), G. Carle, S. Günther, and B. Jaeger, Eds., vol. NET-2020-11-1. Munich, Germany: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich, Nov. 2020, pp. 61–65.

[9] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos framework: A methodology and toolchain for reproducible network experiments," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 259–266.

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[11] Updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://github.com/alpha-unito/streamflow/issues/15

[12] Updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://github.com/nodesource/distributions

[13] T. Dietrich, accessed 2023-03-28]. [Online]. Available: https://gitlab.lrz.de/netintum/teaching/iitm/repos/2023ss-bs/u838.git

Survey of Cryptographic Offloading Techniques for Blockchain Systems

Sebastian Fritsch, Richard von Seck*, Filip Rezabek*,
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: s.fritsch@tum.de, seck@net.in.tum.de, frezabek@net.in.tum.de

***Abstract*—The rise of blockchain systems in recent years has posed challenges to their energy efficiency and scalability. Proposed solutions to these problems have been cryptographic offloading techniques, which we want to present in this paper. Firstly, we introduce blockchain systems based on the blockchain stack and the blockchain trilemma. Secondly, we present the offloading techniques found for each layer and group them into five categories: Proof of Work offloading, Signature Verification offloading, Zero-Knowledge Proof offloading, offloading to the network and Trusted Execution offloading. It was observed that most of the actual offloading of cryptographic operations occure in the context of Proof of Work or Zero-Knowledge Proofs, and despite successful research results, the other offloading categories are often not yet applied in practice and further research is needed.**

***Index Terms*—blockchain, cryptographic offloading, hardware acceleration, FPGA**

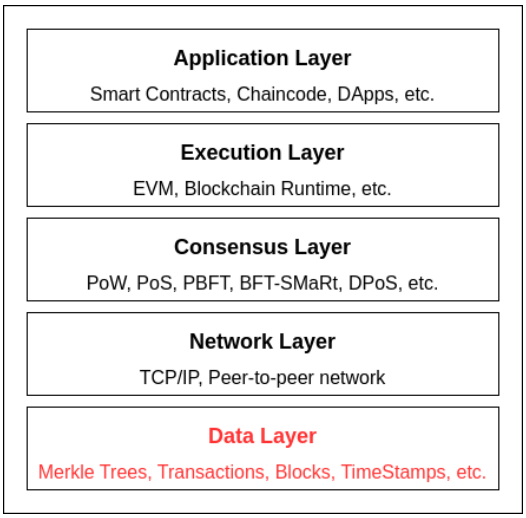


Figure 1: The blockchain stack as presented by [3] [4]

1. Introduction

Blockchain technology has gained immense popularity in recent years for its ability to provide secure and decentralized computation, transfer, and storage of data. Starting with Bitcoin [1] as a proof of concept in 2008, a huge ecosystem of different blockchain systems and technologies has emerged over the last decade.

However, one of the major bottlenecks of further adoption of state-of-the-art blockchain technology into real-world systems is low throughput performance and high latency [2]. To overcome this challenge, researchers have proposed various cryptographic offloading techniques that aim to shift some of the heavy computational tasks from blockchain nodes to other specialized hardware devices.

In this paper we conduct a survey of cryptographic offloading techniques for blockchain systems. It covers various approaches proposed to reduce the computational burden on blockchain nodes, which are categorized into five main categories: Proof of Work (PoW) offloading, Signature Verification offloading, Zero-Knowledge Proof (ZKP) offloading, and Trusted Execution offloading. The objective of this survey is to provide an overview of cryptographic offloading techniques for blockchain systems and to help researchers and practitioners understand the state-of-the-art in this area.

2. Background

Figure 1 shows the blockchain stack as laid out by Fan et al. [3] and Li et al. [4], which provides a basic introduction to the technical side of blockchain systems.

The data layer is the foundational layer of the blockchain, which defines data structures, models, block layout, and storage policies. All higher layers in the blockchain stack use and work on these data structures and in that way the data layer forms a meta-layer, which is symbolized by the red coloring. A very crucial structure are transactions, which contain information signed by a user on how to alter the distributed ledger data in accordance with the runtime rules. Those, in turn, get grouped into blocks, which then get committed together onto the ledger through a consensus mechanism. The exact layout of these data structures varies from one blockchain implementation to another, but they all share some common concepts. For example, the linkage of the different blocks (which coined the term block-chain), through the inclusion of the previous block header in the current block, is essential [4].

A blockchain consists of multiple nodes that communicate in a peer-to-peer network. If everybody can participate in the network, we speak of a public blockchain, otherwise the blockchain system is called permissioned. Similarly to the OSI/ISO model, the network layer facilitates the operation of all higher layers. The network layer is responsible for connections to peers and the propagation of blocks and transactions. The peer-to-peer layout is the basis for the decentralized structure of the blockchain, as there is no machine or network participant that can exclude a peer by itself [3].

 doi: 10.2313/NET-2023-11-1_05

The core of a blockchain is formed by the consensus layer. On this layer, blockchain nodes run an agreed-upon consensus algorithm that decides on the validity of the current state of the ledger and on which blocks to include in it. In the center of most consensus mechanism is a sybil resistance mechanism, that differentiates between trustworthy and non-trustworthy peers, like PoW in Bitcoin [1]. In PoW, miners must find a random value to include in the block so that the block hash has the required number of zero bits. As it is not computationally feasible to solve this efficiently, miners must resort to brute force and invest money in the form of energy to mine the block and "prove their work". In return, miners receive a reward for their efforts, which in most public blockchains is a token share. PoW has been criticized harshly for its vast energy consumption and high latency, so modern blockchains tend to switch to other consensus mechanisms such as Proof of Stake (PoS), Proof of Authority (PoA) or Practical Byzantine Fault Tolerance (PBFT) [2]. Those algorithms do not rely on the computational power of the nodes, but rather on the stake or their identity.

An interesting area of research on novel consensus mechanisms has been in the field of rollups [5]. Rollups attempt to solve the scalability issues of many layer-one chains such as Ethereum by introducing a second layer, that handles the transaction execution and only stores state and proofs on the layer-one chain. We distinguish between two types of rollups: Optimistic rollups and zero-knowledge rollups.

For cryptographic offloading we focus on zero-knowledge rollups, which publish a validity-ZKP to the layer-one to attest the correct execution of transactions. ZKPs are a technology that enable proving the validity of a statement without revealing information about the statement iself. They are also used for applications in Smart Contracts (SCs) such as privacy preserving transfer [5].

The application layer forms the user facing site of the blockchains. Starting with Ethereum in 2015, blockchains enable the execution of SCs and Chaincode on the execution layer [6]. The execution layer offers SCs an execution environment, in which they can interact with blockchain assets and data in ways allowed by the runtime. Examples for such SCs include simple transfers, decentralized exchanges, decentralized autonomous organizations or privacy preserving transfers through ZKPs.

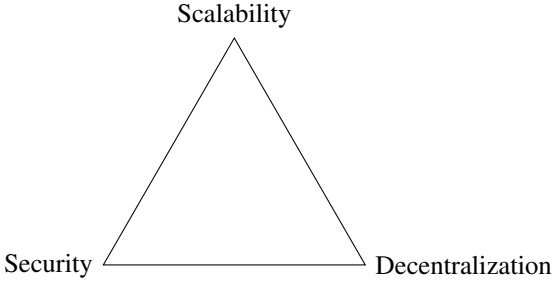


Figure 2: The blockchain trilemma [7]

Figure 2 shows the blockchain trilemma as first presented by Vitalik Buterin [7]. It states that there is an inevitable trade-off between the three properties of Scalability, Security, and Decentralization. As for many public blockchains, security and decentralization are crucial, scalability becomes a major issue. This problem led to more research into scalability solutions like sharding, rollups, or the Lightning Network [4], but also to more attention and research on offloading cryptographic operations [8], which we are going to outline in this work.

3. Overview of Categories

Nearly every layer in a blockchain system involves cryptographic operations, which – depending on their computational effort – may be suitable for cryptographic offloading. In Table 1, we have categorized our findings for cryptographic offloading in five categories and mapped them to the layers, where the offloading technique is used. In the following we are going to outline our findings in detail associated with every blockchain layer.

3.1. Data Layer

Starting with the data layer, we find data structures like Merkle Trees, which are computed using a tree-like structure of hashes. While it is theoretically possible to accelerate these hash computations, we did not find recent research on hardware or network offlading of the data layer.

3.2. Network Layer

Next up with the network layer, some blockchains (especially permissioned blockchains) like Quorum, Ripple and Corda use TLS to secure the data transfer between network participants [29] [30]. Hyperledger Fabric also offers support for TLS, but it is turned off by default [30]. Research by Kim et al. [27] has explored offloading the TLS handshake to SmartNICs, which demonstrated a 5.9x speedup compared to using a CPU. However, there has not been any evidence of adoption of such offloading techniques in blockchain systems.

3.3. Consensus Layer

The consensus layer is home to most cryptographic operations in a blockchain system and therefore designated for offloading them. For the sake of readability we are going to present each offloading category in an own subsection.

3.3.1. Proof of Work Offloading. PoW is a popular sybil-resistance algorithm and the most widespread example of cryptographic offloading to hardware in blockchain systems. The best-known example of this is Bitcoin, which incentivizes block mining through block rewards and transaction fees [9]. In order to mine more correct blocks and save energy, the first Bitcoin GPU miner was introduced in 2010, followed by subsequent developments such as FPGA (Field Programmable Gate Array) miners in 2011 and the first specialized ASIC (Application-Specific Integrated Circuit) miner in 2013 [9]. These advancements led to an increase in mining power from 33 MH/s on Intel Core i7-990x to 13.5 TH/s on an Antminer S9 ASIC. Currently, the two state-of-the-art ASIC miners are

Category	Affected Layers	References
Proof of Work Offloading	Consensus Layer	[9], [10], [11], [12]
Signature Verification Offloading	Consensus Layer, Execution Layer, Application Layer	[13], [14], [15], [16]
Zero-Knowledge Proof Offloading	Consensus Layer, Application Layer	[17], [18], [19], [20], [21], [14]
Trusted Execution Offloading	Consensus Layer, Execution Layer	[22], [23], [24], [25], [26]
Network Offloading	Consensus Layer	[15], [27], [28]

TABLE 1: Categories of Cryptographic Offloading Techniques

BitFury and Bitmain, both of which operate on a 16nm die size and have an energy consumption of about 0.07W per GH/s [9].

Korotkyi and Sachov [12] presented a design for a PoW accelerator for the IOTA cryptocurrency by implementing the Curl hash function on a FPGA, and they were able to achieve a latency speedup of 2100x compared to the software implementation.

On the other hand, efforts have been made to prevent offloading of PoW to specialized hardware by introducing ASIC-resistant algorithms such as multi-hash PoW (e.g., X11, X14), memory-hard PoW (e.g., Ethash, Scrypt, Crypt-Night), or programmatic PoW (e.g., ProgPoW) [31].

Another interesting field of blockchain application is the use in IoT networks. As most IoT nodes do not have access to plentiful energy or computation resources, researchers have proposed offloading the PoW to the edge. This involves taking the computational effort away from the node itself and delegating it to a trusted machine on the edge network [28].

3.3.2. Signature Verification Offloading. As modern blockchains, such as Ethereum, shift their consensus away from PoW due to energy and efficiency concerns, other cryptographic operations become increasingly important. In particular, signature verification was found to be a major performance bottleneck in PoS/PoA chains. In 2020, Toyoda et al. showed that about 5.91 seconds out of 30 seconds blockchain runtime were spent on `crypto.Ecrecover` in a PoA Ethereum blockchain [32]. The function `crypto.Ecrecover` recovers the public key of a given ECDSA signature.

In a work published by Javaid et al. [15], they measured that about 40% of the total execution time on their Hyperledger Fabric blockchain is spent on `ecdsa_verify` and 10% on the SHA256 hashing algorithm. The signature verification is needed on the consensus layer, but also partly on the application layer, as SCs and decentralized Apps (dApps) also often perform cryptographic operations. Therefore, this subsection can also be applied to Section 3.4. These examples signify the optimization speedup in consensus gained by offloading asymmetric crypto operations.

Ikeda [16] published work on offloading ECDSA verification to a FPGA that is twice as fast as a 64-thread AMD EPYC 7601 but uses about 33 times less energy. There is a tradeoff between energy consumption and performance of the implementation and most research we found is focused on energy efficiency. While this is a valid concern, the performance aspect is more important in the context of blockchain systems. A commercial solution offered by SilexInsight and marketed as "Blockchain Hardware Accelerator" [13] focuses on performance and claims 500,000 signature verifications on the `secp256k1` curve per second, which would correspond to a verification time of 2 microseconds. It can be used on FPGAs but is also sold as a proprietary core for ASICs. However, there is no public record of blockchains using the Silex Insight accelerator, and it is unclear whether the 500,000 signatures are measured on a FPGA or an ASIC [13].

Devlin [14] published work on accelerating the Zcash blockchain, focusing not only on ZKP acceleration but also on a signature verification core. He achieved a 1.5x speedup in signature verification with one core and the ability to add more FPGA cores for parallel processing.

The most sophisticated research we found was performed by Javaid et al. [15], who implemented a custom network card on a FPGA that filters blockchain messages and verifies signatures for transactions and blocks, as well as calculates block hashes directly on the FPGA. This design is similar to a SmartNIC, as it can operate on the data before it reaches the CPU. To speed up signature verification itself, they used a propriertary core provided by Mercury Systems. In addition to offloading signature verification, they also accelerated computation of tx and block hashes using "3 stream-based SHA-256 hash calculators" and validation/sanitization of transactions. This enabled a 4.4x speedup in block validation compared to software implementation.

3.3.3. Trusted Execution Offloading. Another interesting area of research has been Trusted Execution Environments (TEEs). TEEs are implemented as a hardware feature by CPU manufacturers and offer a way to protect the confidentiality and integrity of computations that take place inside them. In addition, many TEE vendors offer a way for trusted communication channels to retrieve computing results from the TEE [25]. As the primitives of Trusted Computing are based on cryptography, and the use of them offers ways to accelerate operations, we consider the use of TEEs as a cryptographic offloading technique.

In an overview published by Bao et al. [26], they presented various consensus algorithms that make use of the TEE implementation called "Intel Software Guard Extensions (SGX)". These include Proof-of-Useful-Work (PoUW), Proof-of-Elapsed-Time (PoET), or Secure Proof of Stake (SPoS), which is resistant against the "nothing at stake attack" and the "long-range attack".

3.3.4. Zero-Knowledge Proof Offloading. As explained in Section 2, zero-knowledge proofs are used in the consensus layer for rollup layer-two chains. There are multiple approaches and setups for zero-knowledge proofs. The most famous ones are zkSNARKs, zkSTARKs, and Bulletproofs [33]. At the current time, every known zero-knowledge setup suffers from some performance or decentralization issues, such as creation complexity, validation complexity, proof size, or the requirement for a trusted

setup [34]. This makes it difficult to achieve a performant software implementation. Therefore, multiple researchers have set out to offload the computation of verifications or proofs onto suited hardware.

Weiliang et al. [17] presented GZPK, a proof system for zkSNARKs, that offloads many of the underlying ZKP operations to a GPU. Such underlying functions include Multiscalar Multiplication (MSM) or Number-theoretic transformations (NTT). A similar approach was presented by Lu et al. [18], which focused on the MSM operation. They were able to achieve a speedup of about 2x in comparison to other state-of-the-art GPU implementations. Apart from GPU acceleration, there is research on FPGA offloading. For instance, Peng et al. [21] designed a hardware implementation of the Groth16 zk-SNARK algorithm, which managed to achieve a 10x speedup for creating a proof compared to a reference software implementation. Additionally, some commercial companies are researching on and developing hardware implementations of zero-knowledge proofs, most notably Ingonyama[1] and Cysic[2]. There has also been the "ZPrize"[3] competition in 2022, which focused on accelerating the underlying functions. For example, Ray et al. [20] accelerated the MSM by 15% compared to a FPGA implementation of ZCash, which we reference in Section 3.4. Another prize category was the acceleration of NTT on a FPGA, which was won by team "Supranational". They achieved a performance of 2.47 milliseconds for a transform of the required 2^{24} points [19].

3.4. Execution and Application Layer

The execution and application layers of blockchain systems involve multiple cryptographic operations. For instance, the Solana Sealevel Runtime provides syscalls for cryptographic operations like `sol_secp256k1_recover` or `sol_curve_validate_point` [35]. The Ethereum Virtual Machine (EVM) provides the `ECDSAPUBKEY`, `ECDSASIGN`, and `SNARKV` precompiled contracts [6] that could potentially be offloaded to hardware. Additionally, SCs are free to implement any cryptographic operations they require, leading to a variety of applications. One such application are the ZKPs used for privacy-preserving transfers of digital assets. Tornado Cash and Zcash are two of the most prominent examples of this [36]. Research has been conducted on offloading some of the computations involved in Zcash to FPGAs. For example, Ben Devlin [14] designed a zk-SNARK accelerator by implementing a `bls12-381` coprocessor on a FPGA. He achieved a 2.9x speedup by implementing $\mathbb{F}_p$, $\mathbb{F}_p^2$, $\mathbb{F}_p^6$, $\mathbb{F}_p^{12}$ arithmetic over the `bls12-381` curve, as well as several higher-level operations such as inversion, calculating powers, calculating Frobenius maps, Miller loops, final exponentiation, and optimal ate pairing [14].

Several researchers have proposed leveraging TEEs to execute SCs over private data and to verify the correct execution of these contracts. Bowman et al. [24] employed Intel SGX to achieve these goals for mutually untrusted parties. Meanwhile, Yuan et al. [23] presented ShadowEth, which utilized a similar approach to execute private SCs while preserving existing public blockchains. In this method, only the verification of the correct execution of SCs is performed on the public chain.

In contrast, Das et al. [22] proposed FastKitten, which utilizes TEEs to execute SCs on public blockchains that lack native support for smart contract execution but have a means of storing data. Unlike the previous approaches, FastKitten does not focus on privacy but rather on extending the functionality of blockchains.

4. Discussion

In Section 3, we explored various approaches to offloading cryptographic operations and how they can be applied to different layers of the blockchain stack. We can further categorize these methods based on their scope and impact on the blockchain system.

PoW offloading provides a significant performance improvement in terms of energy costs and speed for the node itself. Without using PoW offloading to ASICs, mining Bitcoin would no longer be profitable due to the high energy costs associated with using GPUs and CPUs [9]. This is likely the reason why PoW offloading is widely used in practice. ZKP offloading to GPUs is currently deployed in Filecoin [4] and there is a lot of research going into offloading to FPGAs and ASICs.

On the other hand, approaches like Signature Verification offloading often only provide a performance gain for the blockchain if the majority of nodes required for consensus adopts them. Otherwise, the blockchain's speed is bottlenecked by the slowest machine required to reach consensus. While there may be potential for energy savings, to our knowledge, this approach has not been widely adopted yet.

While achieving majority adoption of specialized hardware is challenging in permissionless blockchains like Ethereum, there is potential for permissioned blockchains like Hyperledger Fabric. In those settings, the nodes are known and can be required to use a certain hardware or software. This is the focus of the work done by Javaid et al. [15] on Hyperledger Fabric, although they acknowledge that further research and development is needed for the technology and especially for the database connection between specialized hardware and the CPU. This example can be mapped to the blockchain trilemma presented in Figure 2, where one trades decentralization of the blockchain against scalability and security.

Another interesting aspect is the adoption of TEEs in blockchains. Bao et al. [26] have laid out more than 18 different research proposals and projects that make use of Intel SGX. However, there has also been criticism of TEEs. For example, many of these projects are vulnerable to single-point-of-failure attacks, meaning a single compromised SGX enclave could compromise the entire network. Additionally, many solutions depend on "trusted functions", like random number generators, which are provided by the environment and must be trusted by the node operator [26].

1. https://ingonyama.com
2. https://cysic.xyz, https://hackmd.io/@Cysic
3. https://www.zprize.io
4. https://github.com/filecoin-project/rust-fil-proofs

5. Conclusion and future work

In conclusion, offloading cryptographic operations to specialized hardware or TEEs can significantly improve the performance and efficiency of blockchain systems. PoW offloading is already widely used in practice, while other offloading techniques such as signature verification require wider adoption to have a significant impact. Other interesting research has been done on SCs in the Execution and application layer, which can benefit from TEEs and ZKPs for privacy and functionality. Overall, performance and energy improvements are an important step forward but wider adoption and research are needed for some of these techniques.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed: 2023-04-01. [Online]. Available: "https://bitcoin.org/bitcoin.pdf"

[2] A. G. Gad, D. T. Mosa, L. Abualigah, and A. A. Abohany, "Emerging trends in blockchain technology and applications: A review and outlook," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 6719–6742, Oct. 2022. [Online]. Available: https://doi.org/10.1016/j.jksuci.2022.03.007

[3] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.3006078

[4] W. Li, M. He, and S. Haiquan, "An overview of blockchain technology: Applications, challenges and future trends," in *2021 IEEE 11th International Conference on Electronics Information and Emergency Communication (ICEIEC)2021 IEEE 11th International Conference on Electronics Information and Emergency Communication (ICEIEC)*. IEEE, Jun. 2021. [Online]. Available: https://doi.org/10.1109/iceiec51955.2021.9463842

[5] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, vol. 10, pp. 93 039–93 054, 2022. [Online]. Available: https://doi.org/10.1109/access.2022.3200051

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, pp. 1–41, 2022, accessed: 2023-03-30. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[7] V. Buterin, "Why sharding is great: demystifying the technical properties," 2021, accessed: 2023-03-30. [Online]. Available: https://vitalik.ca/general/2021/04/07/sharding.html

[8] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE Access*, vol. 8, pp. 125 244–125 262, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.3007251

[9] M. B. Taylor, "The evolution of bitcoin hardware," *Computer*, vol. 50, no. 9, pp. 58–66, 2017. [Online]. Available: https://doi.org/10.1109/mc.2017.3571056

[10] N. T. Courtois, M. Grajek, and R. Naik, "Optimizing SHA256 in bitcoin mining," in *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2014, pp. 131–144. [Online]. Available: https://doi.org/10.1007/978-3-662-44893-9_12

[11] J. A. Dev, "Bitcoin mining acceleration and performance quantification," in *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, May 2014. [Online]. Available: https://doi.org/10.1109/ccece.2014.6900989

[12] I. Korotkyi and S. Sachov, "Hardware accelerators for IOTA cryptocurrency," in *2019 IEEE 39th International Conference on Electronics and Nanotechnology (ELNANO)*. IEEE, Apr. 2019. [Online]. Available: https://doi.org/10.1109/elnano.2019.8783449

[13] "Silex insight blockchain hardware accelerator product sheet," accessed: 2023-03-30. [Online]. Available: https://www.silexinsight.com/content/uploads/BA452-Blockchain-Hardware-Accelerator_Web.pdf,https://www.silexinsight.com/blockchain-hardware-accelerator/

[14] B. Devlin, "Zcash fpga acceleration engine," 2019, accessed: 2023-03-30. [Online]. Available: https://github.com/ZcashFoundation/zcash-fpga/blob/master/zcash_fpga_design_doc_v1.4.2.pdf

[15] H. Javaid, J. Yang, N. Santoso, M. Upadhyay, S. Mohan, C. Hu, and G. Brebner, "Blockchain machine: A network-attached hardware accelerator for hyperledger fabric," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Jul. 2022. [Online]. Available: https://doi.org/10.1109/icdcs54860.2022.00033

[16] M. Ikeda, "Hardware acceleration of elliptic-curve based crypto-algorithm, ECDSA and pairing engines," in *2021 IEEE 14th International Conference on ASIC (ASICON)*. IEEE, Oct. 2021. [Online]. Available: https://doi.org/10.1109/asicon52560.2021.9620402

[17] W. Ma, Q. Xiong, X. Shi, X. Ma, H. Jin, H. Kuang, M. Gao, Y. Zhang, H. Shen, and W. Hu, "GZKP: A GPU accelerated zero-knowledge proof system," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, Jan. 2023. [Online]. Available: https://doi.org/10.1145/3575693.3575711

[18] T. Lu and L. Peng, "BPU: A blockchain processing unit for accelerated smart contract execution," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Jul. 2020. [Online]. Available: https://doi.org/10.1109/dac18072.2020.9218512

[19] "Accelerating ntt operations on an fpga," 2022, accessed: 2023-03-30. [Online]. Available: https://github.com/z-prize/2022-entries/tree/main/open-division/prize2-ntt

[20] F. Y. Q. Andy Ray, Ben Devlin and R. Yesantharao, "Hardcaml zprize competition," 2022, accessed: 2023-03-30. [Online]. Available: https://zprize.hardcaml.com

[21] B. O. Peng, Y. Zhu, N. Jing, X. Zheng, and Y. Zhou, "Design of a hardware accelerator for zero-knowledge proof in blockchains," in *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pp. 136–145. [Online]. Available: https://doi.org/10.1007/978-3-030-74717-6_15

[22] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," USA, p. 801–818, 2019.

[23] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie, "ShadowEth: Private smart contract on public blockchain," *Journal of Computer Science and Technology*, vol. 33, no. 3, pp. 542–556, May 2018. [Online]. Available: https://doi.org/10.1007/s11390-018-1839-y

[24] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," 2018. [Online]. Available: https://arxiv.org/abs/1807.05686

[25] K. Rabimba, L. Xu, L. Chen, F. Zhang, Z. Gao, and W. Shi, "Lessons learned from blockchain applications of trusted execution environments and implications for future research," in *Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, Oct. 2021. [Online]. Available: https://doi.org/10.1145/3505253.3505259

[26] Z. Bao, Q. Wang, W. Shi, L. Wang, H. Lei, and B. Chen, "When blockchain meets SGX: An overview, challenges, and open issues," *IEEE Access*, vol. 8, pp. 170 404–170 420, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.3024254

[27] D. Kim, S. Lee, and K. Park, "A case for SmartNIC-accelerated private communication," in *4th Asia-Pacific Workshop on Networking*. ACM, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3411029.3411034

[28] S. Wadhwa, S. Rani, Kavita, S. Verma, J. Shafi, and M. Wozniak, "Energy efficient consensus approach of blockchain for IoT networks with edge computing," *Sensors*, vol. 22, no. 10, p. 3733, May 2022. [Online]. Available: https://doi.org/10.3390/s22103733

[29] M. Benji and M. Sindhu, "A study on the corda and ripple blockchain platforms," in *Advances in Intelligent Systems and Computing*. Springer Singapore, Dec. 2018, pp. 179–187. [Online]. Available: https://doi.org/10.1007/978-981-13-1882-5_16

[30] N. Storublevtcev, "Cryptography in blockchain," in *Computational Science and Its Applications – ICCSA 2019*. Springer International Publishing, 2019, pp. 495–508. [Online]. Available: https://doi.org/10.1007/978-3-030-24296-1_39

[31] H. Cho, "ASIC-resistance of multi-hash proof-of-work mechanisms for blockchain consensus protocols," *IEEE Access*, vol. 6, pp. 66 210–66 222, 2018. [Online]. Available: https://doi.org/10.1109%2Faccess.2018.2878895

[32] K. Toyoda, K. Machi, Y. Ohtake, and A. N. Zhang, "Function-level bottleneck analysis of private proof-of-authority ethereum blockchain," *IEEE Access*, vol. 8, pp. 141 611–141 621, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.3011876

[33] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, no. 4, pp. 198–205, Jul. 2021. [Online]. Available: https://doi.org/10.1109/mnet.011.2000473

[34] Y. Gong, Y. Jin, Y. Li, Z. Liu, and Z. Zhu, "Analysis and comparison of the main zero-knowledge proof scheme," in *2022 International Conference on Big Data, Information and Computer Network (BDICN)*. IEEE, Jan. 2022. [Online]. Available: https://doi.org/10.1109/bdicn55575.2022.00074

[35] R. Patel, "Sealevel syscalls," 2022, accessed: 2023-03-30. [Online]. Available: https://bpf.wtf/sol-0x04-syscalls/

[36] T. Chen, A. Lu, J. Kunpittaya, and A. Luo, "A review of zero knowledge proofs," 2021, accessed: 2023-03-30. [Online]. Available: https://timroughgarden.github.io/fob21/reports/r4.pdf

Joint OFDM for Radar and Communication

Thomas Krachten, Leander Seidlitz*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: ge58fag@mytum.de, seidlitz@net.in.tum.de, andre@net.in.tum.de

***Abstract*—The increasing congestion of the radio wave spectrum through the exponentially growing need for wireless communication, combined with the move of communication channels in Radio Detection And Ranging (RADAR) spectrum, lead to the idea of Joint RADAR and Communications (JRC) systems. This paper explores the current state of JRC systems using Orthogonal Frequency Division Multiplexing (OFDM) in the literature. The paper first gives an overview of the concept of OFDM and its basics, an introduction to RADAR systems and the mathematical background of the target detection and tracking with RADAR, followed by the basics of JRC systems and is concluded by a discussion of the currently proposed implementations of JRC systems.**

***Index Terms*—sensing, hight-speed networks, OFDM, RADAR, RadCom, OFDM RADAR, waveform design, FMCW, overview**

1. Introduction

With the increasing demand for wireless communication in the last decades and in the future, the spectrum has and will become more crowded. On top of this, the need for higher transmission speeds means the trend of using higher and higher frequencies, traditionally occupied by Radio Detection And Ranging (RADAR) systems, will continue. This overlap of bandwidth results in in-band and adjacent-band interference creating problems for both applications. [1], [2] Those interferences pose a problem for the communication and military industry but also the aviation, car and space industries due to using RADAR in their respective applications. To solve this problem, the idea of integrating RADAR and communication in one system has surfaced as Joint RADAR and Communications (JRC) or RADAR and Communications (RadCom) system. This integration is possible due to the use of similar components and the move to digital signal processing in both fields. Those components include transmitting and receiving antennas and the signal creation and processing logic. This integration reduces the cost and needed space of the system by reducing the number of components needed but increases the complexity of the JRC system. [3]

2. Frequency multipexing

Multiplexing is a technique that combines multiple signals to one to send over a shared channel to optimize the channel for a chosen criteria, such as data rate. There is a multitude of categories the technique can assigned to,

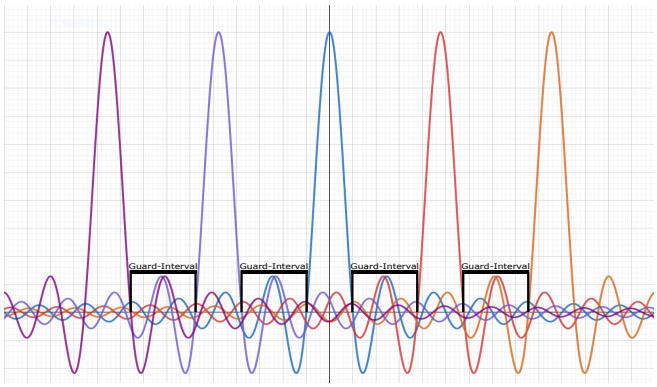


Figure 1: **Diagram in frequency-amplitude-domain of a FDM channel with 5 subbands modulated with rect-functions on. Resulting in 5** $sinc$ **functions with** $sinc(x) = \frac{\sin(x)}{x}$ **with their respective peaks at the carrier-frequencies they are modulated on. The subbands are spaced apart by a guard interval spacing, but there are still Inner Symbol Interference (INSI) between the subbands due to the sidelobes of the** $sinc$**-functions not being zero at the peaks of the other** $sinc$**-functions.**

e.g Time Division Multiplexing (TDM), Space Division Multiplexing (SDM), Frequency Division Multiplexing (FDM) and Code Division Multiplexing (CDM). [4] This paper focuses on FDM and its special form Orthogonal Frequency Division Multiplexing (OFDM) due to its advantages in Joint RADAR and Communications (JRC)-systems.

2.1. Frequency Division Multiplexing

FDM is a technique that divides a channel into multiple none overlapping frequency bands called subbands, which are spaced apart by a guard interval spacing (guard band) as seen in Fig. 1. The subbands have an individual carrier-frequency in the bandwidth of the original channel. A different data stream is modulated on each sub-carrier, e.g. by multiplying the carrier frequency with a sequence of rect-functions representing the bits of the data stream. With the rect-function being defined, in dependency of τ, the duration of the signal being at 1, as:

$$\text{rect}(t) = \begin{cases} 1 & \text{if } |t| \leq \frac{\tau}{2} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The subbands are combined to one signal via an Inverse Fast Fourier Transform (IFFT) and transmitted over the channel to the receiver. The receiver splits the signal

 doi: 10.2313/NET-2023-11-1_06

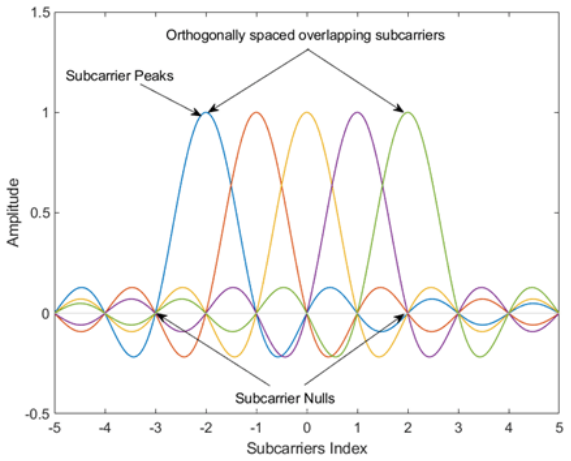


Figure 2: **Diagram in frequency-amplitude-domain of an OFDM channel with 5 subbands modulated with rect-function on. Resulting in 5 $sinc$-functions with peaks at the carrier-frequencies in the frequency-amplitude domain. The subbands are orthogonal, i.e., each maximum amplitude corresponds to the minimum absolute amplitude of the others, by choosing the subcarriers spacing $\delta f = \frac{1}{\tau}$ with τ the length of the window in the rect-function.** [9]

into the subbands with a bandpass filter and demodulates the data streams. [5], [6] The advantages of FDM are the more efficient use of the available bandwidth, no time synchronization is needed and the low complexity of implementations. The disadvantages are the limited number of subbands, the difficulty in assigning the frequencies to the subbands and the INSI between subbands. The INSI stems from the sidelobes of the $sinc$-function not being zero at the peaks of all the other functions. Therefore, adding or subtracting from the combined amplitude of the signal at that frequency. [7] To overcome these disadvantages, OFDM was developed.

2.2. Orthogonal Frequency Division Multiplexing

OFDM is a digital modulation form of FDM in which the subbands are orthogonal to each other. This means the adjacent sub-carriers do not interfere with each other because the maximum power of each sub-carrier corresponds directly to the minimum power of all the other sub-carriers as seen in Fig. 2. [8] This orthogonality is archived by using sine (or cosine) waves with frequencies of $k \cdot \Delta f$ with $k \in \mathbb{Z}$ and $\Delta f = \frac{1}{\tau}$, where τ is the window length of the rect-function.

Through this, the distance between two adjacent carrier frequencies can be controlled by choosing a fitting τ and the need for guard-bands is eliminated. Therefore, higher spectral efficiency is archived compared to FDM. However, due to multipath propagation and the possibility of dispersion in the frequency domain, most OFDM systems use a Cyclic Prefix (CP) to reduce the ISI as seen in Fig. 3. The CP is a part of the signal that is repeated at the beginning of the next signal and is used to compensate for the delay of the channel. The length of the CP is chosen to be longer than the maximum delay of the channel. The receiver removes the CP, and the signal is processed as if it was not delayed. [10]

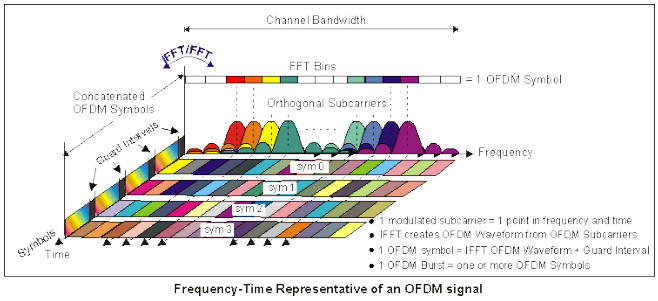


Figure 3: **Diagram in frequency-amplitude-time-domain of an OFDM channel with 8 orthogonal subbands modulated with rect-function on combined to OFDM-symbols. Between the OFDM-symbols in the time domain guard interval are inserted to reduce Inter Symbol Interference (ISI).** [11]

The carrier-frequencies have data streams with a certin maximum length modulated onto, are then combined into one signal via an IFFT. This OFDM symbol is transmitted over the channel to the receiver. The receiver then uses a Fast Fourier Transform (FFT) to split the signal into the sub-carriers and demodulates the data streams. [11] The advantages of OFDM are the high spectral efficiency, resistance to multipath propagation and no ISI. OFDM comes at the cost of increasing the complexity of the implementation.

3. Radio Detection and Ranging

This section gives an overview of the basics of Radio Detection And Ranging (RADAR) systems and the calculations needed to determine the angle, distance and velocity of a target. It also gives an overview of the different types of RADAR systems and their waveforms.

3.1. Basics of RADAR

A RADAR system uses electromagnetic waves in the frequency range of 3MHz to 100GHz to detect targets in its range. A pulse, the transmitted pulse (TP), is transmitted via an antenna and the echo, the received pulse (RP), is processed to determine the target's angle, distance and angular velocity. The RP can vary from the TP in frequency and amplitude. The components of a RADAR system vary, but all have at least one oscillator to create the TP, one or multiple antennas and a logic to process the RP. If only one antenna is used for transmission and receiving, a RADAR system is called monostatic, quasi-monostatic if transmitting and receiving antennas are close to one another and bistatic if the antennas are at different locations from the target's viewpoint. On top of the angle, range, and velocity sensing functions, RADAR systems can also detect the size of the target, shape, material and moving parts. However, the complexity of RADAR systems and their cost and size increase with the number of functions they can perform. [12]

RADAR waveforms. There are two basic options for a waveform when designing a RADAR system: a Continuous Wave (CW) or a pulse. With a CW RADAR system, the TP is a constant wave, and the receiver is continuously active. Therefore, the RP can only be separated from the current TP through a frequency change induced by the movement of the transmitter, target or receiver (Doppler

shift), mixing the CW with the received signal or via spatial separation of transmitter and receiver. The pulse RADAR transmits a pulse and then waits for the echo. Both intervals together are called Pulse Repetition Interval (PRI). [12], [13]

Classification of RADAR systems. RADAR systems can be classified in different ways. One way is to classify them by the way they transmit the TP. This can be done by classifying them in CW and pulse RADAR systems. Another way is to classify them by the amount of transmit and receive antennas, if there are multiple receive antennas and one transmit antenna the system is called Single Input Multiple Output (SIMO) and when there are multiple transmit and receive antennas Multiple Input Multiple Output (MIMO). [13]

3.2. The RADAR equation

Every signal transmitted from an antenna or received by one is damped by a certain factor, depending on the used antenna. This is the result of its gain. Every antenna has a specific amount of gain G, that is its radiation efficiency η multiplied by the directivity D. As a result, the power of the RP depends on the power of the signal before transmission, the gain, the wavelength and the distance to the target and its RADAR-cross-section. The equation to determine this power is given by $P_r = \frac{P_s G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4}$ with P_r denoting the power of the RP in W, P_s the power of the original signal in W, G_t the gain of the transmitting and G_r the gain of the receiving antenna, σ the RADAR-cross-section of the target in m^2, λ the wavelength of the TP and R the distance to the target. The RADAR-cross-section is the area of the target TP illuminates. It can be reduced by using RADAR-wave dampening materials or with an optimized shape, e.g. fewer 90-degree angles. The power is also dampened by the distance to the target with a factor of the distance to the power of four due to the quadratic dampening of the TP over the one-way distance. Due to the dampening of the TP by multiple factors, the echo of a target might be too weak that its Signal to Noise Ratio (SNR) is so low that the echo is not distinguishable from the noise, making it undetectable. To combat this, the signal is filtered with filters matched to the TP and possibly integrated between pulses (adding up the magnitudes from multiple echoes) increasing the SNR, if the PRI is small enough that the target has only moved a negligible distance. [12]

Clutter detection and suppression. Clutter is any unwanted signal echo that is not from a target, e.g. the echo of a rock face or a bird. The simplest way to detect clutter is by comparing the RP to the RPs of previous cycles. If the echo is similar to the echoes in RPs of previous cycles with similar range, angle and velocity in relation to the movement of the RADAR system during the cycles, the echo is considered clutter. It is also possible to use the staticity of most clutter to detect it. To suppress detected clutter, its echo in the RP can be ignored, or the average of previous echoes can be subtracted from the current echo, and the threshold for target detection can be raised resulting in fewer false positives.

3.3. Range detection

To estimate the distance to a target, the time between the transmission of the TP and the reception of the RP is needed. As a result, CW RADAR needs to assign the echo to an earlier point in time to determine the range. This can be archieved via frequency modulation, e.g. increasing the frequency of the TP with time and dropping back to the base frequency after a certain time t_c. This interval is called a chirp, and the system is called Frequency Modulated Continuous Wave (FMCW) RADAR. The range formula for (quasi-)monostatic PRI systems is $R_t = \frac{cT_R}{2}$, with c being the speed of light, R_t the distance between the transmitter and the target and T_R the time between transmitting the pulse and receiving it the round trip time which can be measured. In contrast, bistatic RADAR PRI systems need a synchronization element to determine the delay. This can be done via a reference channel if the distance between transmitter and receiver is known. The range formula for bistatic systems is $R_t + R_r = cT_R$, with R_t and R_r being the distance between the transmitter and the target and the receiver and the target. The noise and interference in real-world applications adds some ambiguity to the range detection. Furthermore, if the range exceeds the maximum unambiguous range $R_{maxu} = \frac{c \cdot PRI}{2}$ [14] the target seems to be closer than it is. This is called range ambiguity. A target might not be detected at all due to overlapping echoes. This occurs when there are multiple targets in the same direction and with a distance smaller than the range resolution R_{res}. For pulse RADAR systems the $R_{res} = \frac{c \cdot PRI}{2}$ [15] and for FMCW systems the $R_{res} = \frac{c}{2B} T_c f_s$ [16], with B being the bandwidth of the FMCW, T_c the duration of the cycle and f_s the sampling rate.

3.4. Radial velocity detection

As mentioned before, the RP will be shifted in the frequency domain through any distance change between transmitter and target, and target and receiver. This Doppler shift will be positive if the distance is decreasing and be negative if it is increasing. If the Doppler shift induced by the target is known, the radial velocity of the target can be calculated with $v_r = \frac{f_d \lambda}{2}$. If it is unknown, it can be estimated with the range of the target at multiple pulses. For a target moving faster than the maximum unambiguous Doppler velocity $v_{max} = \frac{\lambda}{4 \cdot PRI}$ an exact velocity can not be calculated. On top of that, when the target is moving at the radial velocity of $n \cdot v_{blind} = \frac{\lambda}{2 \cdot PRI}$ with n being an integer or f_d being an integer multiple of $\frac{1}{PRI}$, the target appears not to be moving at all. This is called blind speed. [12], [17]

3.5. Angle detection

With an isotropic antenna, i.e., the antenna emits equally in all directions, only the range and velocity of the target can be detected. This can be resolved by using a directional antenna and moving the antenna in azimuth (horizontally) and elevation (vertically) mechanically or by using beamforming. Thus, the angle of the target can be estimated to be in the direction of the transmit beam. Better accuracy of angle detection can be archieved by using multiple receive antennas spaced apart by $d_r = \frac{1}{2}\lambda$. Those receive the same signal, but through the space between them, the signal travels an additional distance of $d_r \sin\theta$. This results in a phase shift (see Fig. 4). This phase shift ω has to be measured so the angle of arrival θ can be calculated with $\theta = \sin^{-1}\left(\frac{\omega\lambda}{2\pi d_r}\right)$. Due

 doi: 10.2313/NET-2023-11-1_06

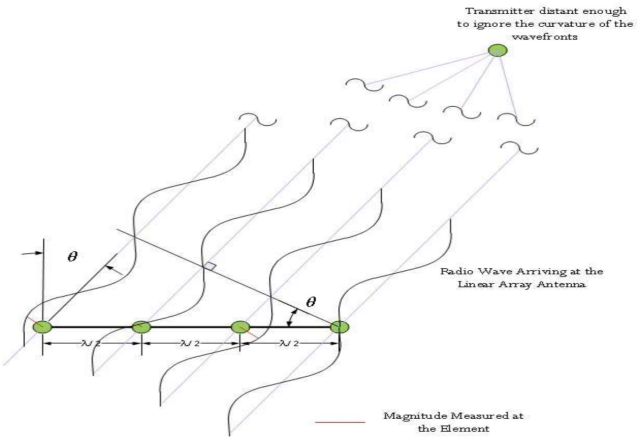


Figure 4: **Schematic of a RADAR system with 4 receiving antenna spaced apart $\frac{\lambda}{2}$ with λ being the wavelength. The received wavefront appears linear through the distance to the source at the angle θ. This angle results in a different time of arrival and therefore with a shifted phase.** [18]

to the fact that ω can only be estimated in the range of $(-\pi, \pi)$, the unambiguous Field Of View (FOV) is limited to $\theta_{fov} = \pm \sin^{-1}\left(\frac{\lambda}{2d_r}\right)$. If more antennas are used ω can be estimated more reliable, and a FFT can be performed on the signal sequence. The peaks of the result indicate the angle of arrival. Increasing the number of receiving antennas N_r leads to sharper peaks, thus higher accuracy. If the antennas are spaced apart by the distance d_r the angle resolution is $\theta_{res} = \frac{2}{N}$ with $N = N_t \cdot N_r$, if the transmitting antennas are spaced apart by the distance $d_t = N_r \cdot d_r$. This positioning allows the detection of N_t different transmission, which increases the sensing capabilities of the RADAR system. [13]

4. RADAR and communication systems

JRC or RADAR and Communications (RadCom) systems combine RADAR and communication systems. The idea of JRC systems is to share the components of the RADAR, such as transmitting and receiving antennas, signal generators and signal processing logic. JRC systems also have the upside of sharing information between the subsystems, which can be used to enhance both, e.g. by using the RADAR to detect communication partners and using beamforming to increase the range and SNR of the communication system. The targets are to reduce the interference between the RADAR and communication system, cost and space. There are two main approaches to JRC systems, the first is to use the TP of the RADAR to send data to a communication partner and to detect targets in one pulse. The second approach is to switch between the TP for radar and for communication dynamically, depending on the need for communication. Of course, FMCW JRC systems can be used, but the transfer rates are currently not high enough. A combination of the two is feasible but loses the ability to sense continuously, as well as other downsides. [16]

Possible applications. There are many areas that could profit from JRC systems. The two focus points are intelligent transport systems, e.g. self driving cars or adaptive cruise control, and the military. JRC systems could enable cars to communicate with each other about their properties like position, speed and predicted route and with infrastructure like traffic lights or sensors in the road through protocols like Vehicle to Everything (V2X). This should be archivable, while not making the car dependent on market penetration, making it more reliable and secure by its sensing capabilities and attractive to car manufacturers. The military could use JRC systems to detect and track targets and to communicate with allied assets and command centers while reducing the cost of acquisition space and power needed for two separate systems. [19]

Waveform. Designing a wave that satisfies prerequisites for both RADAR and communication is the main challenge of JRC systems. Stand-alone RADAR systems use waves specifically designed for a high autocorrelation, i.e., the similarity between the wave at one time and at a later point in time. [20] Using regular RADAR waves would result in orders of magnitude lower symbol rates compared to a communication system in the same bandwidth. One option to decrease the need for autocorrelation in a wave, is using multicarrier waveforms such as OFDM. Those also solve the problem of the low transfer rates of normal RADAR waves. Additionally to the need for a specific autocorrelation and bit rate, there might be other requirements for the wave, such as certain sensing capabilities, a low Bit Error Rate (BER), low probability of being detected or increased resistance to jamming. All those properties need to be considered when designing a JRC waveform. OFDM has many of the most commonly needed properties, such as a high bit rate, resistance to multipath fading, low BER and cost of implementation, due to its extensive use in communication systems and the resulting availability and low cost of components. [19]

5. Current State of RadCom Systems

The current state of JRC systems is mainly theoretical, with only prototypes and simulations created to prove some of the proposed concepts. This is because the JRC systems are still in the research phase and are not yet ready for commercial use. This section will give an overview of current proven implementations of JRC systems in scientific papers. The primary focus lies in designing the waveform, as it is the main challenge of JRC systems. This, combined with the different demands of the use cases, results in many propositions for waveform designs for JRC systems. However, due to the need for a high data rate in most RadCom systems, a particular focus is put on OFDM-based waveforms, as they are known to achieve high data rates.

Shared OFDM subchannels. One of the more unique suggested solutions is using precoded OFDM symbols in a MIMO RADAR system. The precoder is designed to minimize a specific sensing-communication metric and can be used for beamforming. On top of that, the OFDM subchannels are divided into two groups, private and shared. A sub-carrier is shared when all transmitting antennas in the RADAR system can transmit on it, enabling high data transmission rates. The parallel transmission on the same sub-carrier leads to the carriers losing their orthogonality and creates coupling between the transmitted symbols.

The loss of orthogonality complicates the estimation of the target properties by preventing the formation of a virtual array. The number of private sub-carriers and the assignment to a transmission antenna to increase the accuracy of the RADAR system is dynamically decided on, depending on the situation. The private sub-carrier can be used to create a virtual array and for pilot transmission for channel estimation, but they decrease the archivable transfer rates. [1]

Non-contiguous OFDM subbands. Another proposition is to use non-contiguous OFDM subbands for data transmission located in a large spectrum for sensing. However, using regular OFDM waves has the downside, of a high Peak to Average Power Ratio (PAPR) resulting in lower amplifier efficiency, and in-band and out-of-band distortion. To combat this, the sensing bands need to transmit waves with good autocorrelation, optimized with a unique algorithm to decrease the PAPR of the entire spectrum. [21]

Downsides of OFDM-based waveforms. The downsides of OFDM-based waveform are mainly the same as normal OFDM has, combined with the high PAPR. However, the main disadvantage of OFDM-based waveforms in JRC systems is the reduction of SNR compared to traditional RADAR systems, resulting in lower accuracy of the target parameters estimations, probability of detection and higher BER. This effect can be reversed to within a margin by transmitting the OFDM symbol multiple times until the next symbol and using specific pilot symbol patterns and modulation schemes to decrease the BER. However, the sending of symbols continuously makes currently widely used access patterns like Carrier Sense Multiple Access (CSMA) harder to implement. [16]

6. Conclusion and future work

This paper has given an overview of the basics of FDM, OFDM, RADAR and JRC, and the current state of RadCom systems. It shows that OFDM is a good candidate for RadCom systems due to its high data rates and the possibility of a low PAPR. However, the use of OFDM in RadCom systems is not without downsides. The use of normal OFDM in RadCom systems does not reach the possible maximum transmission rates. To reach these rates, extra steps in the signal creation and processing are needed. Overall, we believe that OFDM is the best candidate for a JRC waveform due to the availability and low cost of components and their for JRC systems' favorable properties. In the future, more research has to be done in the field of RadCom waveform design, their implementation, robustness to jamming, multipath fading and beamforming. Furthermore, protocols might be needed for the communication part of the JRC system as some propositions are making it impossible to use currently widely used access control protocols.

References

[1] Z. Xu and A. Petropulu, "A bandwidth efficient dual-function radar communication system based on a mimo radar using ofdm waveforms," *IEEE Transactions on Signal Processing*, vol. 71, pp. 401–416, 2023.

[2] B. D. Cordill, S. A. Seguin, and L. Cohen, "Radar performance degradation with in-band ofdm communications system interference," pp. 1–1, 2013.

[3] e. a. Xiao, B., "Ofdm integrated waveform design for joint radar and communication." *Wireless Networks (2023)*, 2023.

[4] Elektonik Kompendium. Multiplex und multiplexing. Accessed on 03.26.2023. [Online]. Available: https://www.elektronik-kompendium.de/sites/kom/0211292.htm

[5] ——. Fdm - frequency division multiplex (frequenzmultiplexverfahren). Accessed on 03.26.2023. [Online]. Available: https://www.elektronik-kompendium.de/sites/kom/2305151.htm

[6] R. Awati. frequency-division multiplexing (fdm). Version of 08.2021. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/frequency-division-multiplexing

[7] Geeks for Geeks. Frequency division and time division multiplexing. Accessed on 03.25.2023. [Online]. Available: https://www.geeksforgeeks.org/frequency-division-and-time-division-multiplexing/

[8] National Instruments. Ofdm and multi-channel communication systems. Accessed on 01.06.2022. [Online]. Available: https://www.ni.com/de-de/innovations/white-papers/06/ofdm-and-multi-channel-communication-systems.html

[9] MathWorks. ofdmmod. Accessed on 03.27.2023. [Online]. Available: https://de.mathworks.com/help/comm/ref/ofdmmod.html

[10] S. B. Thomas Strohmer. Optimal ofdm design for time frequency dispersive channels. Version of 07.2003. [Online]. Available: https://ieeexplore-ieee-org.eaccess.tum.edu/document/1214833

[11] Keysight Technologies, Inc. Concepts of orthogonal frequency division multiplexing (ofdm) and 802.11 wlan. Accessed on 03.26.2023. [Online]. Available: https://rfmw.em.keysight.com/wireless/helpfiles/89600B/WebHelp/Subsystems/wlan-ofdm/content/ofdm_basicprinciplesoverview.htm

[12] P. D. Jenn. Radar fundamentals. Accessed on 03.28.2023. [Online]. Available: https://faculty.nps.edu/jenn/seminars/radarfundamentals.pdf

[13] S. Rao. Mimo radar. Version of 7.2018. [Online]. Available: https://www.ti.com/lit/ml/swra554a/swra554a.pdf?ts=1679955023785&ref_url=https%253A%252F%252Fwww.google.com%252F

[14] A. M. Society. maximum unambiguous range. Version of 04.25.2012. [Online]. Available: https://glossary.ametsoc.org/wiki/Maximum_unambiguous_range

[15] D. Koks. How to create and manipulate radar range-doppler plots. Version of 12.2014. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA615308.pdf

[16] K. B. S. A. Dapa, G. Point, S. Bensator, and F. E. Boukour. (2023) Vehicular communications over ofdm radar sensing in the 77 ghz mmwave band.

[17] Dav University. Blind speed and staggered prf notes final. Accessed on 03.29.2023. [Online]. Available: https://www.davuniversity.org/images/files/study-material/Blind%20Speed%20and%20staggered%20PRF_notes%20final.pdf

[18] mathscinotes. Beamforming math. Accessed on 03.30.2023. [Online]. Available: https://www.mathscinotes.com/2012/01/beamforming-math/

[19] C. Sturm and W. Wiesbeck, "Waveform design and signal processing aspects for fusion of wireless communications and radar sensing," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1236–1259, 2011.

[20] Pennsylvania State University. Autocorrelation and time series methods. Accessed on 04.01.2023. [Online]. Available: https://online.stat.psu.edu/stat462/node/188/

[21] Y. Huang, S. Hu, S. Ma, Z. Liu, and M. Xiao, "Designing low-papr waveform for ofdm-based radcom systems," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6979–6993, 2022.

Current State of Hardware and Algorithms in WiFi Radars

David Pop, Leander Seidlitz*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: {popda,seidlitz,andre}@net.in.tum.de

***Abstract*—Recent advances in wireless sensing demonstrate the ability of commodity WiFi waves to detect human activity. Using IEEE 802.11 WiFi instead of cameras, wearable sensors and LiDARs can be a widely available and cost-effective solution to the human surveillance and detection problem. It can play a crucial role in applications, such as healthcare, intrusion detection systems and smart homes.**

WiFi sensing schemes rely on using the fine granularity of the physical layer CSI made possible by the OFDM modulation technique. This paper introduces the necessary theoretical background to cover wireless sensing. The most relevant state of the art is analyzed to determine the performance of WiFi based on different metrics and scenarios. Recent works are able to accurately estimate human pose using deep learning approaches. WiFi-based human recognition proves to be a reliable detection mechanism using off-the-shelf and low-cost hardware.

***Index Terms*—wifi, ofdm, csi, human detection**

1. Introduction

Human activity detection and sensing has been analyzed for various surveillance and monitoring applications for years. Such solutions are widely applicable in domains such as intrusion detection systems [1], healthcare [2], smart home [3], monitoring of children and elderly [1] or even augmented reality [3] and gaming [4].

When considering approaches for the estimation of human positioning, there are mainly three solutions. The first option relies on approaches based on visualization [5], such as 3D cameras. In this case, the cameras are used to capture images of humans to then recognize and match their activities [3]. However, the performance of this approach can be altered by physical factors, e.g., lighting conditions or angles blocked by objects [3]. A second solution to the detection problem is the use of wearable sensors [5], such as LiDAR and radar sensors [6]. Nonetheless, such devices have to be attached to the targets of the detection, which is not always feasible. Both of these approaches are characterized by high costs, power and energy consumption, and may sometimes be out of reach for daily use. This work focuses on another detection method which uses 1D sensors, more specifically, commodity WiFi signals for sensing.

The use of wireless signals for activity recognition has been receiving attention due to its cost-efficiency and high availability. Initial research has used the received signal strength indicator (RSSI) for indoor localization [7]. However, using the physical (PHY) layer for *channel state information (CSI)* proves to be more accurate for indoor localization than RSSI [7]. This is due to CSI's ability to have multipath characteristics, which offers a more accurate representation of the environment [7]. Multiple detection approaches use the PHY layer CSI provided by WiFi. The accuracy of this solution relies on the fine granularity of channel information offered by the *orthogonal-frequency-division-multiplexing (OFDM)* modulation, which is the standard for IEEE 802.11 a/g/n WiFi standards [7], [8]. For this approach, any activity in the environment is detected using the amplitude and phase information of the OFDM sub-carriers. The use of WiFi for indoor detection can be more effective than its counterparts since it is not prone to errors caused by illumination, dead angles and it also offers an off-the-shelf solution [6]. When comparing it to wearable sensors, it is also "non-invasive", since it is not required that the targets wear any additional equipment [1]. Thus, using commodity WiFi signals to detect human activity through walls can be a cost-effective, widely available and more private approach.

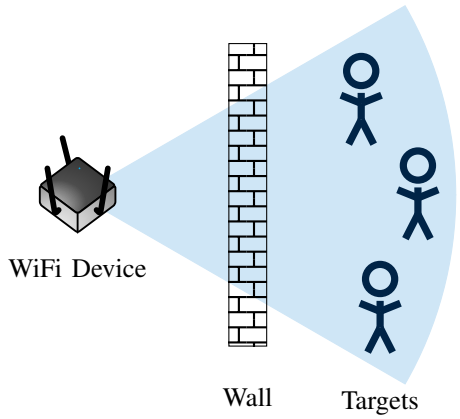


Figure 1: Environment for through-wall detection of human activity using WiFi signals.

Figure 1 displays a typical system for through-the-wall detection using standard IEEE 802.11 WiFi. The blue highlighted area represents the sent WiFi signals coming from the transmitter antennas. Note that in this case, the WiFi device stands for both transmitter and receiver. The radio frequency (RF) signal crosses the wall and reflects off objects and targeted humans [4]. The detection of human activity is realized by capturing and analyzing these reflections. The main challenge is to distinguish

 doi: 10.2313/NET-2023-11-1_07

between reflections that generate from the wall itself and from other objects in the room from the actual target. The power of the received signal is also highly reduced, since it has to pass through the wall twice [4]. The initial strong reflection from the wall creates a "flash effect" [4], which can hinder the sensing of the environment behind the wall. The CSI shows the correlation between transmitted and received signal waves, making it possible to detect motion in the environment [6].

The paper is organized as follows. Relevant properties of the used signal processing techniques, such as OFDM and CSI are described in 2. Section 3 shows the common approaches to the through-wall detection and sensing problem by presenting relevant state of the art. Finally, the conclusion and key findings are given in Section 4.

2. Preliminaries

This section offers background information needed to determine how WiFi signals are used to detect and sense humans through surfaces. The underlying multiplexing technique used by WiFi is the OFDM. Furthermore, the PHY layer CSI uses information offered by OFDM to detect motion between transmitted and received signal waves. Therefore, the two are introduced below.

2.1. Orthogonal Frequency-Division Multiplexing

OFDM is a multiplexing technique used in wireless applications, such as the IEEE 802.11 WiFi [8]. The main principle of OFDM is to split a high volume of data into smaller parts which are then transmitted simultaneously through sub-carriers. This makes OFDM a multi-carrier system [9], meaning that the channel is transformed into a set of multiple orthogonal carriers, which do not interfere with each other. The total bandwidth from the spectrum is split into multiple bands corresponding to each carrier, making it possible to transmit data in parallel [10]. In OFDM, the sub-channels are able to overlap without having interfering frequency spectra at the peak of the subband due to the orthogonality [8], given by the following condition:

$$\int_0^T \cos(2\pi n f_0 t)\cos(2\pi m f_0 t)\,\mathrm{d}\,t = 0, \quad n \neq m \tag{1}$$

where $n, m \in \mathbb{Z}_{\neq 0}$, f_0 is the fundamental frequency and T the time period of the integration [10]. Furthermore, one sub-carrier signal can be described as follows [8]:

$$s_n(t) = a_n e^{j2\pi f_n t} \tag{2}$$

for the transmitted data $\{a_0, a_1, ..., a_{N-1}\}$ and carrier frequency f_n. The sum of these signals of the N sub-carriers represents the whole sent signal, which corresponds to the following equation [8]:

$$s_k = \sum_{n=0}^{N-1} a_n e^{\frac{j2\pi nk}{N}} \tag{3}$$

Using the discrete Fourier transform (DFT) on the signal defined in 3, the received data can be recovered [8].

Figure 2 depicts a typical OFDM system, containing an OFDM transmitter and receiver part. The input

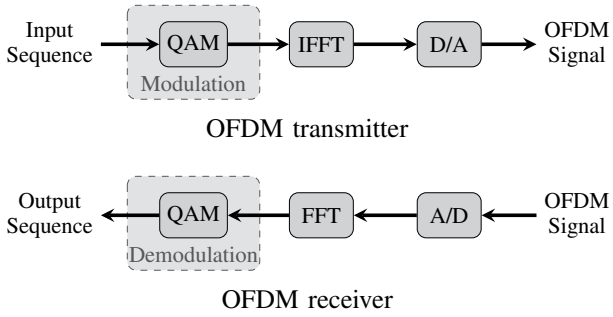


Figure 2: OFDM system based on [8], [10].

information sequence is modulated using quadrature amplitude modulation (QAM), thus modulating the OFDM subcarriers. The signals are then transformed using inverse fast Fourier transform (IFFT). The resulting OFDM signal is completed after being transported through a digital-to-analog (D/A) converter. On the receiving end, the process is similar, but uses an analog-to-digital (A/D) converter, FFT and is demodulated to create the resulting output data.

In a real OFDM application, for IEEE 802.11n WiFi, a 20 MHz bandwidth centered around a 2.4 GHz or 5 GHz central frequency is used [3]. Depending on the scenario, 30 up to 64 sub-carriers can share the channel. The bandwidth and sub-carrier number may also deviate.

2.2. Channel State Information

The first step towards human activity detection using wireless signal relies on the CSI given by the PHY layer. CSI describes the relation between transmitted and received signal wave [6]. Previously, the MAC layer received signal strength indicator (RSSI) was mainly used for wireless detection. Together with the sub-channel information of multiple-input-multiple-output (MIMO) and OFDM in IEEE 802.11 WiFi, CSI is able to deliver finer-grained information of the environment [5]. Being able to use the phase and amplitude information of each OFDM sub-carrier makes it more suitable and performant than its data-link layer counterpart [7].

In a WiFi channel, for each sub-carrier, transmitting a signal $\boldsymbol{x}$ and receiving a signal $\boldsymbol{y}$ denotes to:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n} \tag{4}$$

$$H_i = |H_i|e^{j\angle H_i} \tag{5}$$

where $\boldsymbol{H}$ is the CSI matrix and $\boldsymbol{n}$ the noise vector [11]. The CSI matrix estimates the modulated activity in the environment given the WiFi waves [5]. The three dimensions for the complex CSI matrix are for the i-th subcarrier with N_T transmitter and N_R receiver antennas [5]. Furthermore, the amplitude $|H_i|$ and phase $\angle H_i$ for each complex CSI value H_i of a sub-carrier can be denoted as in equation 5 [5], [7].

3. WiFi Detection Implementations

There are multiple approaches possible to the WiFi sensing problem. This Section gives an overview of the most relevant state of the art in the domain. Works that do

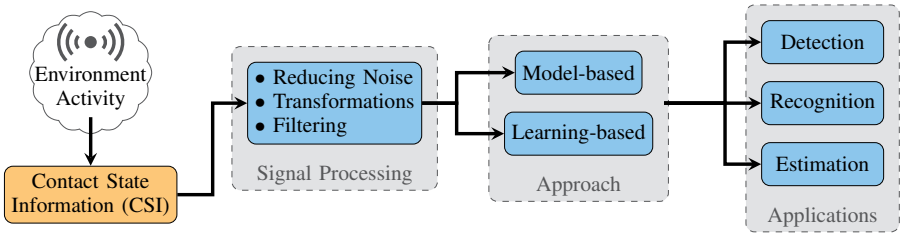


Figure 3: Approaches for through-wall human sensing using WiFi based on [3], [5], [11].

not necessarily cover through-wall detection play an important role in the ongoing research and are thus analyzed as well. Table 1 gives an overview of the main characteristics of the works, such as accuracy scores, used hardware and covered human positioning during the experiments. Even though there are multiple implementations, most of them follow a similar pattern. Figure 3 depicts a block diagram of the typical flow when it comes to human detection using WiFi waves. The first step is to generate the WiFi signals to capture environment activity. The fine granularity of CSI is used to extract information from the received signal. The captured CSI measurements are then processed by applying noise reduction, transformations (e.g., FFT, discrete wavelet transform) and filtering techniques to eliminate outliers and increase performance [11]. Processed CSI can be used by a model-based or learning-based approach for different applications, such as human detection, recognition or estimation.

Passive Bistatic WiFi Radar (PBWR). In [12], Chetty et al. present one of the first attempts (2012) for through-the-wall (TTW) detection with WiFi by using a passive bistatic WiFi radar. The authors test their implementation to detect a moving human. The WiFi wave signal transmitter is a 802.11 WiFi AP placed in the same room, 4 m away from the target. The passive bistatic radar consists of two receivers placed outside the room at a standoff distance. The data is processed offline first by applying range-Doppler mapping. The *CLEAN algorithm* is then introduced to remove digital signal interference (DSI) and additional stationary clutter. For target detection, a 2D constant false alarm rate is used. The signal-to-interference ratio (SIR) is the main metric for the TTW target responses. Results show that using CLEAN, the SIR is decreased by 19 dB, creating a more accurate detection.

Wi-Vi. Adib and Katabi [4] present Wi-Vi, a TTW detection device. The main components of the Wi-Vi device are two transmitter and one receiver antennas. Compared to the previous implementation, Wi-Vi does not need to have any device located within the same room as the target. One main aspect analyzed by this work is the initial reflection from the wall that is much stronger than the reflections off the objects behind the wall, creating a "flash effect". The authors use iterative nulling together with power boosting to tackle this challenge, by nulling the strongly reflected signal. An inverse synthetic aperture radar (ISAR) is used for motion tracking. Experiments are carried within a conference building having walls of different building materials and thickness. Wi-Vi can detect one or multiple moving human targets and gestures. The detection scores are 100% for 0 to 1 targets. For multiple humans, Wi-Vi shows an accuracy of 85%. When considering gesture detection, the accuracy reaches 93% for closer and 75% for longer standoff distances. Wi-Vi delivers high detection accuracy for thin building materials (wood, glass, door, 15 cm hollow wall), but drops at 87% for a concrete 20 cm wall.

DeMan. Wu et al. introduce DeMan [1], a solution for "non-invasive DEtection of moving and stationary huMAN with commodity WiFi devices". Unlike previous works, DeMan also focuses on detecting stationary humans by choosing breathing (chest motions) as a sensing factor. However, this work does not cover through-wall detection. The scheme is based on the amplitude and phase values of the OFDM sub-carriers given by CSI. The experimental hardware consists of a IEEE 802.11n WiFi AP transmitter and a laptop equipped with a NIC receiver. DeMan delivers high accuracy for the conducted experiments in true positive scenarios: 99.86% for absent, 93% for stationary and 95% for moving humans respectively.

WiSpy. Hanif et al. propose WiSpy [13], a CSI-based through-wall movement sensing and person counting scheme, which uses commodity WiFi waves. Two Intel NUCs are placed in front of a 33 cm brick wall. In their approach, the authors use machine learning (ML) algorithms on the processed CSI data to predict the amount of people behind the wall. Principal component analysis (PCA) is applied on the CSI measurements for dimensionality reduction. This work also compares multiple ML algorithms on the PCA data. Results show, that using decision trees (DT) delivers the best results, i.e., a detection accuracy of 96.97%. On the other hand, k-nearest neighbor (KNN) delivers the poorest performance, having a detection accuracy of under 80%.

Person-in-WiFi (PiW). In [14], Wang et al. introduce one of the first WiFi-based person perception schemes, implementing body segmentation and pose estimation. A deep learning approach is used to map WiFi samples to 2D human body segmentation using recorded RGB videos. During the experiments, subjects are placed between transmitter and receiver, without having any walls or obstructing stationary objects in-between. The setup consists of two WiFi NICs, one used for transmission, one for receiving, each having 3 antennas. To implement a deep learning approach for person perception, CSI measurements and video frames are taken at the same time stamps. Body segmentation maps are constructed using region-based convolutional neural networks (R-CNN). For pose estimation, however, the Body25 model of OpenPose is used. The proposed approach shows high performance

TABLE 1: WiFi radar sensing implementations comparison.

	PBWR [12]	**Wi-Vi** [4]	**DeMan** [1]	**WiSpy** [13]	**PiW** [14]	**HARNN** [3]	**DensePose** [6]	**GoPose** [15]
General								
accuracy	–	>85%	>94%	96.97%	>85%	>95%	>87%	<5cm error
carrier freq.	2.4 GHz	2.4 GHz	2.4 GHz	5.18 GHz	2.4 GHz	5 GHz	5 GHz	5.32 GHz
bandwidth	16 MHz	20 MHz	20 MHz	20 MHz	20 MHz	20 MHz	20 MHz	40 MHz
hardware	DWL 2000AP+	WiFi antennas	Intel 5300 NIC	Intel NUC	Intel 5300 NIC	Intel 5300 NIC	WiFi antennas	Intel 5300 NIC
Human position								
moving	✓	✓	✓	✓	✓	✓	✓	✓
stationary	✗	✗	✓	✗	✗	✓	✓	✓
pose estimation	✗	✗	✗	✗	✓	✗	✓	✓
through-wall	✓	✓	✗	✓	✗	✗	✗	✓

for whole person profiles, having detection scores between 85% and 91%. Low performance of WiFi signal detection is demonstrated for small body parts due to WiFi's wavelength of 12.5 cm.

HARNN. A CSI-based WiFi detection scheme for human activity recognition using recurrent neural networks (HARNN) is introduced by Ding et al. in [3]. This work firstly uses a two-level decision tree leveraging the variance and correlation coefficient of the CSI measurements to detect activity in the environment. Moreover, the noise is eliminated from the raw CSI using channel power variation (CPV) and time-frequency analysis (TFA) in time and frequency domain respectively. Detection of various human activities (e.g., running, walking, sitting etc.) is achieved using a RNN model with a long short-term memory (LSTM) block. A WiFi device is used for transmission together with a Intel 5300 NIC as a receiver. The experiments are carried in a closed off environment, through-wall detection, however, is not covered. HARNN reaches detection accuracies of 95% and 96% for all the mentioned activities. The authors also point out, that increasing the amount of receivers yields better detection rates of 96% up to 98% on average.

DensePose. Geng et al. propose DensePose [6], a study that aims to achieve body segmentation and key-point body detection using commodity WiFi signals. Similar to HARNN, after sanitizing the raw CSI data, the amplitude and phase of the 30 OFDM sub-carriers are mapped to 2D feature maps. The data is passed through a DensePose-RCNN architecture, used to predict UV coordinates of the human body. UV maps create a correlation between 3D and 2D human data. The main goal here is to map 1D CSI data to UV maps, thus transforming the data into spatial domain. The correlation between CSI samples and video captures is achieved similar to HARNN. In addition, DensePose uses transfer learning from the image-based network to the WiFi-based one to reduce training time. The testing environment uses three transmission and three emission antennas. The authors test out their approach in multiple layout scenarios. The implementation yields high accuracies of over 87% for the same layout used in training. However, when deployed within an unknown layout, the average precision (AP) of the model drops (e.g., from 43 AP to 27 AP). The detection accuracy also suffers when faced with human body poses, which did not occur during training. Moreover, the results are not entirely clear once there are more than three human targets in the testing space. The authors thus motivate generating more training data to solve the failure cases.

GoPose. In [15], Ren et al. present GoPose, a scheme used to estimate human pose using WiFi signals. The novelty relies on the tracking of 3D skeleton-based human poses, compared to the 2D version of PiW. The implementation is able to track both stationary and moving targets. Unlike previous works, GoPose manages to estimate unseen activities as well. It also works when being faced with walls, screens or other stationary objects. The scheme builds up on sanitized CSI measurements of 30 OFDM sub-carriers. In addition, it uses the 2D angle of arrival (AoA) spectrum to determine between reflections off objects and targeted bodies. To map 2D AoA spectra to 3D skeletons of humans, the authors use a deep learning approach based on CNN and LSTM. Estimating the 3D pose of people requires a higher amount of devices for the setup. One transmitter and four receivers are used in the testing environment. The transmitter uses three linear antennas, whereas the receivers are equipped with L-shaped antennas. Results are evaluated using joint localization errors. GoPose is able to accurately track stationary human targets, having low errors of 0.4 cm. Testing through-wall detection yields errors of an average 4.7 cm. Similar to prior works, the estimation success rate decreases when faced with multiple people due to multiple reflections.

4. Conclusion

This paper analyzes the use of commodity WiFi signal waves for human sensing. Preliminaries required for the standard IEEE 802.11 WiFi signal analysis, such as CSI and OFDM, are introduced. The analyzed related works show that WiFi radar is a competitor to cameras or wearable sensors due to its wide availability, power efficiency and low cost. Under normal circumstances, most implementations reach detection accuracies over 85%.

With the rise of machine learning over the years, previous CSI-based sensing schemes have been improved using more complex deep learning architectures to be able to estimate 3D posing of humans. Current research still faces issues when it comes to detecting smaller body parts, large stand-off distances and multiple human targets. Moreover, the proved accurate WiFi sensing also raises an issue to the networking community regarding security and regulations of WiFi signals.

References

[1] C. Wu, Z. Yang, Z. Zhou, X. Liu, Y. Liu, and J. Cao, "Non-Invasive Detection of Moving and Stationary Human with WiFi," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2329–2342, 2015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7102722

[2] Y. Wang, K. Wu, and L. M. Ni, "WiFall: Device-Free Fall Detection by Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 2, pp. 581–594, 2017. [Online]. Available: https://www.researchgate.net/publication/316561246_WiFall_Device-Free_Fall_Detection_by_Wireless_Networks

[3] J. Ding and Y. Wang, "WiFi CSI-Based Human Activity Recognition Using Deep Recurrent Neural Network," *IEEE Access*, vol. 7, pp. 174 257–174 269, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8918311

[4] A. Fadel and K. Dina, "See Through Walls with Wi-Fi!" *ACM SIGCOMM'13*, 2013. [Online]. Available: https://people.csail.mit.edu/fadel/papers/wivi-paper.pdf

[5] Y. He, Y. Chen, Y. Hu, and B. Zeng, "WiFi Vision: Sensing, Recognition, and Detection with Commodity MIMO-OFDM WiFi," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8296–8317, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9076313

[6] J. Geng, D. Huang, and F. De la Torre, "DensePose From WiFi," *arXiv preprint arXiv:2301.00250*, 2022. [Online]. Available: https://arxiv.org/abs/2301.00250

[7] Z. Yang, Z. Zhou, and Y. Liu, "From RSSI to CSI: Indoor Localization via Channel Response," *ACM Comput. Surv.*, vol. 46, no. 2, dec 2013. [Online]. Available: https://doi.org/10.1145/2543581.2543592

[8] S. B. Weinstein, "The history of orthogonal frequency-division multiplexing [History of Communications]," *IEEE Communications Magazine*, vol. 47, no. 11, pp. 26–35, 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5307460

[9] D. Tse and P. Viswanath, "Fundamentals of wireless communication." USA: Cambridge University Press, 2005, ch. 3, pp. 35–101. [Online]. Available: https://web.stanford.edu/~dntse/wireless_book.html

[10] M. Bhardwaj, "A Review on OFDM: Concept, Scope & its Applications," *IOSR Journal of Mechanical and Civil Engineering*, vol. 1, pp. 07–11, 01 2012. [Online]. Available: https://www.iosrjournals.org/iosr-jmce/papers/vol1-issue1/B0110711.pdf

[11] Y. Ma, G. Zhou, and S. Wang, "WiFi Sensing with Channel State Information: A Survey," vol. 52, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3310194

[12] K. Chetty, G. E. Smith, and K. Woodbridge, "Through-the-Wall Sensing of Personnel Using Passive Bistatic WiFi Radar at Standoff Distances," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 4, pp. 1218–1226, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6020778

[13] A. Hanif, M. Iqbal, and F. Munir, "WiSpy: Through-Wall Movement Sensing and Person Counting Using Commodity WiFi Signals," in *2018 IEEE SENSORS*, 2018, pp. 1–4. [Online]. Available: https://asif-hanif.github.io/_pages/SENSORS2018.pdf

[14] F. Wang, S. Zhou, S. Panev, J. Han, and D. Huang, "Person-in-WiFi: Fine-Grained Person Perception Using WiFi," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 5451–5460. [Online]. Available: https://arxiv.org/abs/1904.00276

[15] Y. Ren, Z. Wang, Y. Wang, S. Tan, Y. Chen, and J. Yang, "GoPose: 3D Human Pose Estimation Using WiFi," vol. 6, no. 2, jul 2022. [Online]. Available: https://doi.org/10.1145/3534605

Prediction of Rare Latency Events

Leonard Scheerer, Max Helm*, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: leonard.scheerer@tum.de, helm@net.in.tum.de, jaeger@net.in.tum.de

***Abstract*—An increasing number of safety critical applications rely on networks and their latency bounds. Thus, providing estimates of worst case latencies is crucial for ensuring the quality-of-service requirements of such systems. This paper presents an approach to acquire these estimates using Extreme Value Theory, a statistical method that bases its predictions on models derived from real-world measurements. In particular, we analyze the latency values of a single flow of a virtualized network topology. Additionally, we compare the approach with alternative methods and present current applications of Extreme Value Theory in the networking area. We consider Extreme Value Theory a powerful tool for estimating the tail-end of network latency distributions. In many cases, it outperforms alternative approaches with similar goals.**

***Index Terms*—extreme value theory, latency measurement**

1. Introduction

Safety-related systems, such as those in medical, aerospace and security fields are known to be *time-critical*. That is to say, missing a deadline can have drastic consequences on the environment, on equipment or even human lives. More and more of these critical systems are now distributed and therefore rely on networks. One prominent example are vehicular networks that need to exchange real-time status updates between individual vehicles. For this reason, ultra-reliable and low-latency communication is a key service type in the next generation (6G) communication systems. Realizing networks with low end-to-end latency guarantees represents a major challenge facing 6G [1] [2].

This paper utilizes Extreme Value Theory (EVT) to model the extreme latency events of a single network flow. The raw latency data originate from an experiment conducted by Wiedner et al. The authors make use of real networking hardware to create a virtualized network topology on a single physical host. The detailed measurement setup is described in [3]. All code used to visualize and analyze the described data is available online at https://github.com/leonardscheerer/rare-latency-events.

The remainder of this paper is structured as follows: Section 2 introduces the theoretical background of Extreme Value Theory. In Section 3, various approaches to the prediction of extreme events are discussed. Section 4 applies EVT to real-world latency data. After presenting other applications of EVT in the networking area in Section 5, some concluding remarks are made in Section 6.

2. Background

This section introduces basic concepts and results in the field of Extreme Value Theory.

2.1. Extreme Value Theory

EVT is a branch of probability theory with the aim of describing the stochastic behavior of extremes, i. e., events on exceptionally large or small scales. The derived models are a solid theoretical basis for predicting the occurrence and extent of rare events and enable extrapolations to unobserved levels. This goal is unique amongst the statistical disciplines, as commonly, the objective is to model the ordinary, rather than the unordinary [4].

EVT had its first applications in the 1950's in the area of civil engineering, in which the frequency and magnitude of natural phenomena such as floods or earthquakes can be crucial information for the design of structural components of buildings [4]. In recent years, EVT has gained considerable traction in various other fields such as the social sciences, the medical profession, economics and even astronomy [5].

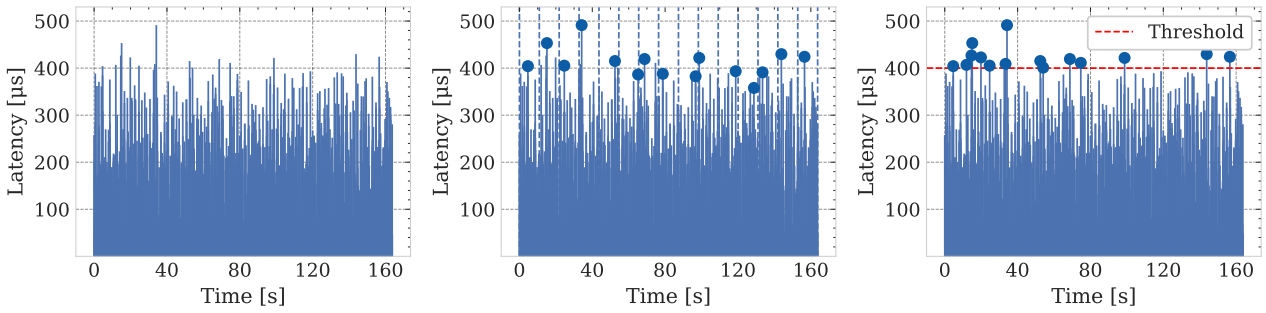
Conforming to the *extreme value paradigm*, the extrapolations used to predict extreme values are based on asymptotic arguments, i. e., on using mathematical limits as finite-level approximations. As a consequence, the results of EVT cannot be regarded as exact when applied to finite samples [4, Chapter 1].

Before trying to model the behavior of extreme events, it is first necessary to define what constitutes an extreme occurrence. There are two main ways to define such events, leading to two alternative methods of mathematical modeling [6]. Both approaches, termed the Block Maxima approach and the Peaks over Threshold approach, are used in practice and are briefly explained in the following two sections.

2.2. Block Maxima Approach

In the Block Maxima approach the observation period is divided into n non-overlapping blocks of equal size. The maximum of each block is deemed to be an extreme value. Figure 1a shows the raw latency data and Figure 1b highlights the extreme values when applying the Block Maxima approach with $n = 15$. When interested in exceptionally small events, extreme values are derived analogously with minima instead of maxima.

 doi: 10.2313/NET-2023-11-1_08



(a) Raw data (b) Block Maxima approach (c) PoT approach

Figure 1: Latency data of a single network flow and their extreme values according to the Block Maxima and PoT approach.

The Block Maxima are subsequently modeled using the *Fisher-Tippett-Gnedenko* theorem. It states that under certain assumptions, mainly that these maxima are samples of independent and identically distributed random variables, the distribution of the maxima converge to one of three probability distributions: the Gumbel Distribution, the Fréchet Distribution or the Weibull Distribution. The distributions are also referred to as the extreme value type 1, type 2 and type 3 distributions, respectively [6].

All three distributions can be represented with a single distribution, the Generalized Extreme Value Distribution (GEV). The cumulative distribution function $F(x;\mu,\sigma,\xi)$ of the GEV is given by (1). It measures the probability that the random variable will take a value less than or equal to x.

$$F(x;\mu,\sigma,\xi) = \exp(-\max\{1+\xi\frac{x-\mu}{\sigma},0\}^{-\frac{1}{\xi}}) \quad (1)$$

ξ is termed the extreme value index and maps to the aforementioned three distributions. To derive a robust model, ξ as well as the scale parameter σ and location parameter μ have to be fitted to the observed data using a suitable estimation method [5, Chapter 4].

2.3. Peaks over Threshold Approach

In the Peaks over Threshold (PoT) approach, we specify some threshold u. All values that exceed this threshold are considered extreme values. Figure 1a shows the raw latency data and Figure 1c highlights the extreme values when applying the PoT approach with $u = 400\,\mu s$. When interested in exceptionally small events, extreme values are derived analogously with threshold subceedances instead of threshold exceedances.

The obtained excesses, i. e., the amounts that the peaks exceed the threshold, follow the *Pickands-Balkema-De Haan* theorem. It states that with a sufficiently high threshold and under similar conditions to the Fisher-Tippett-Gnedenko theorem, the values of the excesses will converge to the Generalized Pareto Distribution (GPD). The cumulative distribution function $G(x;\sigma,\xi)$ of the GPD is given by

$$G(x;\sigma,\xi) = 1-\max\{1+\frac{\xi x}{\sigma},0\}^{-\frac{1}{\xi}} \quad (2)$$

Similarly to (1), ξ determines the shape and σ the scaling of the distribution. Equation (2) does not contain a location parameter, as it is fixed to the previously chosen threshold [5, Chapter 4] [7].

3. Analysis

In this section we argue that dedicated methods are necessary to accurately predict extreme behavior. Afterwards we analyze selected modeling approaches for extremes, particularly in the context of latency events.

3.1. Traditional Methods

Traditional parametric statistical methods are ill-suited to model values at the very tail-end of a distribution. These statistical methods typically aim to be a good fit for a large proportion of the observed data, thus, accurately representing regions where most of the data fall. However, this comes at the price of a worse fit in the tails and therefore justifies the usage of dedicated approaches. Nevertheless, separate methods for modeling extreme values such as EVT are not needed – and possibly not suited – for estimating values that make up the top 10 %, 5 % or perhaps even 1 %. Rather, these methods focus on *extreme* (e. g. 0.1 %) outliers [8].

3.2. Modeling Approaches

Machine Learning. One possible approach to predicting rare latency events is machine learning. This method has emerged as a fast and reliable means to data-driven predictions. Wambura et al. [9] propose using a deep neural network for real-time stochastic extreme events prediction. The authors empirically confirm that their approach is fast and accurate. Their experimental results also suggest superior performance compared to well-known prediction methods. Nevertheless, application of deep learning to latency events is not straightforward and requires a large number of training samples due to a slow convergence in the training phase. Low learning efficiency can be combatted by the integration of knowledge of the environment such as estimated packet loss [10].

Network Calculus. Another possible approach to modeling and preventing high latencies are provable worst case upper latency bounds. This can be achieved

via network calculus, a system theory for communication networks. The theoretical framework is built on the non-traditional min-plus and max-plus algebras [11]. Network Calculus and similar formal methods work on simplified assumptions and do not incorporate environmental events such as electromagnetic interferences. Additionally, the derived bounds are not tight [12].

Extreme Value Theory. The remainder of this paper focuses on applying EVT to the prediction of rare latency events. As discussed in Section 2, both the Block Maxima as well as the PoT approach are used in practice. The Block Maxima approach particularly lends itself to modeling data sets that already consist of block maxima, e. g. records of annual maximum sea-levels. In this case, the approach can incorporate all of the measurements and formulate accurate predictions. In practice, however, it is uncommon to have data of this form and following the Block Maxima approach may entail a wastage of information. Suppose, for example, that there are several recorded high events during one block. The block maximum takes precedence over all other events of a block. They are ignored as a consequence of this approach – even if they were noteworthy in the sense that they exceed the Block Maxima of other blocks [4]. This is not the case for the threshold exceedances in the PoT method. For this reason, the PoT approach is considered to utilize extreme observations more efficiently than the Block Maxima approach [13]. Note that both the maxima as well as the threshold excesses are assumed to be independent of each other in the respective theorems described in Section 2.2 and 2.3. Whereas this is often a reasonable assumption in the Block Maxima approach, as they are spaced out by construction, this cannot be said for the PoT approach. Thus, the PoT approach is often used in combination with special techniques such as declustering that aim to ensure that the data are independent [4, Chapter 5]. Based on the aforementioned benefits and drawbacks of the Block Maxima and PoT approaches, we deem the PoT technique to be more suitable for the characterization of the tail distribution of latencies.

4. Rare Latency Estimation

This section applies Extreme Value Theory to the real-world latency data introduced in Section 1 in order to predict rare latency events. We use the PoT approach described in Section 2.3 and assume that the raw data consist of a sequence of independent and identically distributed measurements.

4.1. Inference

To fit the generalized Pareto family to the observations, we first select a suitable threshold and subsequently estimate the characterizing scale parameter σ and shape parameter ξ.

Threshold Selection. Threshold selection is a crucial part of extreme value analysis following the Peaks over Threshold method. Too low a threshold is likely to lead to the Generalized Pareto Distribution not being a good fit for the threshold excesses, as a sufficiently high threshold is a requirement of the Pickands-Balkema-De Haan theorem. Too high a threshold results in very few exceedances – and thus less information – for the estimation of the model. One tool for the selection of an appropriate threshold is the mean residual life plot. Figure 2 shows the mean residual life plot of the latency data and its approximate 95 % confidence intervals based on the approximate normality of sample means.

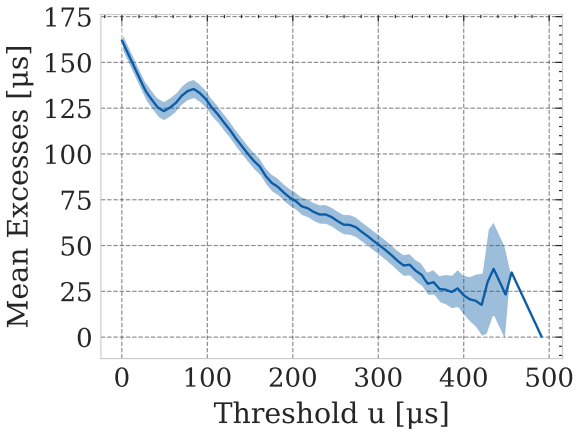


Figure 2: Mean residual life plot for the latency data of a single network flow.

The mean residual life plot depicts the average excess value over the given threshold for a set of different values of the threshold u. According to [4, Section 4.3.1], the mean residual life plot should be approximately linear in u above a threshold u_0 at which the GPD provides a valid approximation for the threshold excesses. In practice, the interpretation of the mean residual life plot often proves to be difficult as it involves a great deal of subjective judgement. Based on Figure 2, we decide to use a threshold of $u = 370\,\mu s$ because of the approximate linearity for from $u = 370\,\mu s$ to $u = 425\,\mu s$. This leads to 42 threshold exceedances, a proportion of about 3.32 %. It might be tempting to suggest a higher threshold such as $u = 425\,\mu s$ as there is some evidence for a linear relationship. However, this would result in only 4 exceedances, too few for a meaningful inference. Similarly, lower thresholds provide an excessive number of exceedances violating the asymptotic assumption of Extreme Value Theory.

Parameter Estimation. There are several fit methods to derive the parameters of (2). Using maximum likelihood estimation, we get

$$(\sigma, \xi) \approx (27.813, -0.064) \tag{3}$$

The 95 % confidence intervals for σ and ξ are [16.141, 39.485] and [−0.356, 0.227], respectively. We omit technical details and simply refer to [7, Chapter 3] and [4, Section 4.3.2].

4.2. Model Checking

Model checking consists of assessing the quality of a fitted generalized Pareto model based on plots and different metrics. In this paper, we focus on probability

plots as a graphical technique to evaluate the quality of the parameter estimates in (3). In practice, however, various other means such as quantile plots, return level plots or density plots can also be useful to determine the goodness-of-fit of a model. The probability plot for the fitted model is shown in Figure 3.

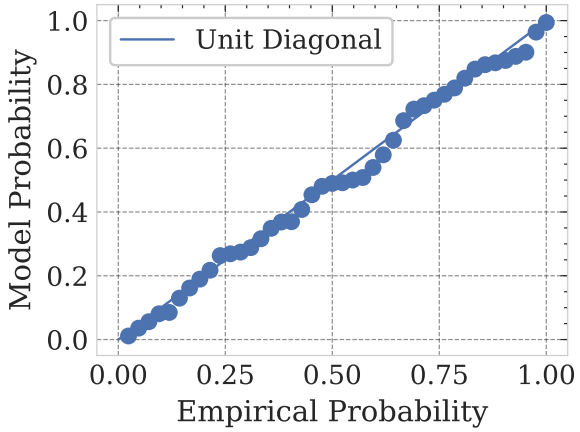


Figure 3: Probability plot for the latency data of a single network flow.

In general, probability plots are a tool for assessing the degree to which a data set follows a given distribution. In Figure 3, the data are plotted against the GPD with the model parameters of (3). The construction of probability plots ensures that the points should lie close to the unit diagonal if the data set follows the given distribution [4, Section 4.3.5]. No substantial departures from linearity can be seen, so that we deem our model to be suitable for extrapolation.

4.3. Extrapolation

After having estimated suitable parameter values, it is possible to utilize the derived model to predict extreme values. Usually, it is convenient to interpret extreme value models in terms of return periods and return levels. The former, the return period, corresponds to the average time between extreme events. The latter, the m-observation return level, describes the value that is expected to be exceeded exactly once in the next m observations [4, Section 4.3.3].

Using the model parameters of (3), the 10 000-observation return level is calculated to be approximately 505 µs, i. e., a latency value above 505 µs is expected to be witnessed only once every 10 000 observations. Assuming that the number of network packets per second stays constant at 7.7 packets/s, 10 000 observations correspond to about 22 min. Note that this is an extrapolation to an unobserved level. The data set only consists of 1264 observations sent over a period of 2.7 min with a maximum latency of about 491 µs.

The 95 % confidence interval is calculated to be approximately [429 µs, 580 µs] via the Delta Method. The confidence intervals often tend to be large, as uncertainty can be magnified in extrapolation.

5. Applications

This section is dedicated to presenting applications of Extreme Value Theory in the networking area.

CBA-EVT. Wang et al. [14] propose to use EVT in a medium access control (MAC) protocol designed for battery-powered wireless sensor networks (WSNs). WSNs are networks of spatially dispersed sensors that monitor physical conditions of the environment. Amongst other areas, they are used in earth sensing, e. g. for natural disaster prevention. It is paramount that MAC protocols for WSNs are energy-efficient to ensure that the sensors can serve their intended functions longer. CBA-EVT is such a MAC protocol that aims to be energy-efficient while also avoiding long latencies. It is named after the two theoretical methods that are the foundation of the protocol: Cost Benefit Analysis and Extreme Value Theory. For a given time slot, Extreme Value Theory in CBA-EVT is used to estimate the completion time of each node, i. e., the time after which no further packets will have to be received in this time slot. This can be used to enable the node to enter a low-power mode early during one time slot and thus saving energy without sacrificing latency.

Vehicular networks. Extreme Value Theory is also used to ensure stringent latency and reliability constraints in vehicular networks. Vehicle-to-vehicle safety applications are inherently time-critical, as individual vehicles rely on acquiring real-time status updates from each other. One commonly used metric is the age of information (AoI). It measures the time elapsed since the latest status update that reached its intended destination has been generated at its source. As argued by Abdel-Aziz et al. [2], minimizing the average AoI in vehicular networks cannot fulfill the unique requirements of ultra-reliable and low-latency vehicular communication. Instead, the authors use Extreme Value Theory to reduce the probability of outliers in the AoI distribution and show the achieved improvements of their approach with simulation results.

Wireless networks. Vehicular networks are a special case of wireless networks. As shown by Mouradian [12], Extreme Value Theory is particularly attractive for studying worst case delays in wireless networks. In contrast to wired networks, wireless networks are more susceptible to unpredictable behavior of the environment such as electromagnetic interference. These disturbances cannot be captured by formal methods like network calculus. As a result, statistical methods, especially Extreme Value Theory, are a valuable tool for the study of worst case delays in wireless networks.

6. Conclusion and Future Work

In this paper we discussed different prediction approaches for rare latency events. In particular, we looked at a general statistical method called Extreme Value Theory and the two main approaches therein: the Block Maxima approach and the Peaks over Threshold approach.

To investigate the Peaks over Threshold method in greater detail, we modeled the tail-end of the latency distribution of a single network flow. The latency data originated from a network experiment on a single physical host using real networking hardware. We consider this approach a powerful tool not only for estimating worst case latencies but also for many other applications in the networking area. However, the accuracy of the predictions is limited by the quality and amount of measurement data and by the assumptions about the data.

To reduce the complexity of the analysis, we assumed that the measurements are independent and identically distributed. The assumption of independence has already been relaxed by Helm et al. [15]. Future work can explore further relaxation of these assumptions and their effect on the accuracy and reliability of the predictions. The consequences of following the Block Maxima approach instead of the PoT method in the context of predicting high latencies also require further investigation.

References

[1] S. R. Pokhrel, J. Ding, J. Park, O.-S. Park, and J. Choi, "Towards enabling critical mMTC: A review of URLLC within mMTC," *IEEE Access*, vol. 8, pp. 131 796–131 813, 2020.

[2] M. K. Abdel-Aziz, C.-F. Liu, S. Samarakoon, M. Bennis, and W. Saad, "Ultra-reliable low-latency vehicular networks: Taming the age of information tail," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[3] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[4] S. Coles, J. Bawa, L. Trenner, and P. Dorazio, *An introduction to statistical modeling of extreme values*. Springer, 2001, vol. 208.

[5] J. Galambos, "Extreme value theory for applications," in *Extreme Value Theory and Applications: Proceedings of the Conference on Extreme Value Theory and Applications, Volume 1 Gaithersburg Maryland 1993*. Springer, 1994, pp. 1–14.

[6] M. I. Gomes and A. Guillou, "Extreme value theory and statistics of univariate extremes: a review," *International statistical review*, vol. 83, no. 2, pp. 263–292, 2015.

[7] L. Haan and A. Ferreira, *Extreme value theory: an introduction*. Springer, 2006, vol. 3.

[8] F. X. Diebold, T. Schuermann, and J. D. Stroughair, "Pitfalls and opportunities in the use of extreme value theory in risk management," *The Journal of Risk Finance*, vol. 1, no. 2, pp. 30–35, 2000.

[9] S. Wambura, H. Li, and A. Nigussie, "Fast memory-efficient extreme events prediction in complex time series," in *Proceedings of the 2020 3rd International Conference on Robot Systems and Applications*, 2020, pp. 60–69.

[10] C. She, C. Sun, Z. Gu, Y. Li, C. Yang, H. V. Poor, and B. Vucetic, "A tutorial on ultrareliable and low-latency communications in 6G: Integrating domain knowledge into deep learning," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 204–246, 2021.

[11] A. Van Bemten and W. Kellerer, "Network calculus: A comprehensive guide," 2016.

[12] A. Mouradian, "Extreme value theory for the study of probabilistic worst case delays in wireless networks," *Ad Hoc Networks*, vol. 48, pp. 1–15, 2016.

[13] A. Bücher and C. Zhou, "A horse race between the block maxima method and the peak–over–threshold approach," *Statistical Science*, vol. 36, no. 3, pp. 360–378, 2021.

[14] T.-L. Wang, J.-C. Kao, and S. A. Ciou, "CBA-EVT: A traffic-adaptive energy-efficient MAC protocol for wireless sensor networks," in *2014 Wireless Telecommunications Symposium*. IEEE, 2014, pp. 1–6.

[15] 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)*. IEEE, 2022, pp. 359–363.

Digital Twins of Computer Networks

Martin Tonauer, Kilian Holzinger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: m.tonauer@tum.de, holzinger@net.in.tum.de

***Abstract*—Usage of a digital twin (DT) is already an established technology in various industries like aerospace, construction or traffic analysis. However, in the field of computer networks its utilization is still rare. Although already in usage for example in the roll-out of 5G networks, the true potential of a digital twin network (DTN) has yet to be developed.**

This paper tries to contribute a concise overview of DTN technology not only by giving an insight into the background of this technology and some of the nomenclature but also by summarizing properties and requirements for a functioning DTN. The paper also remarks the problems surrounding the implementation of a working DTN. To help in mitigating these, it offers a simple conceptual implementation model to promote further research into designing DTNs. The paper also presents current and possible future use cases as well as open questions for further attention.

***Index Terms*—Digital Twin (DT), Digital Twin Network (DTN)**

1. Introduction

The concept of a digital twin (DT) has been used in different types of industries for the purpose of simulating real world hardware, applications or whole scenarios without interfering with the physical system (PS) itself. Born out of the desire to monitor, test or experiment on these systems in a safe environment, DTs have seen widespread usage in many industries, from automotive production all the way to weather forecasting [1] [2]. Given the broad range of utilization it would seem logical for DTs to have a similar share in computer networks, but the potential and abilities of such a digital twin network (DTN) are only starting to be valued [3]. A DTN in the scope of this paper is the simulated counterpart to a real and physical computer network. The goal of this essay is to give the reader an overview of this technology and outline some applications and their benefits to designing, constructing and maintaining computer networks.

The rest of this paper is structured as follows: Section 2 will give a brief overview of the historical background and motivate the usage of DTs, while Section 3 will define the terminology that is used in this paper. Section 4 introduces properties and requirements of DTs and DTNs. A possible model of a DTN is presented in Section 5 with some application examples following in Section 6. Section 7 discusses the shown concepts and corresponding open questions. A concluding summary of this paper can be found in Section 8.

2. Background and Current Status

Having a first, second or third draft of something is probably as old as human craftsmanship. But what happens when the desired product becomes final and goes into production? With increasing complexity of systems, the need to test or experiment before actually committing new features to a real world product becomes ever more important. Being able to verify that the addition of new machines to an existing production line will not unintentionally alter the functionality or output of the line before the actual installation of the hardware is an important capability of a production plant. Similarly, testing a new software patch on a real satellite while it is in orbit can have drastic repercussions maybe even to the point where it can no longer communicate. Therefore, the roll-out of this patch and the correct subsequent operation of the satellite have to be checked beforehand. Using a stand-in is an intuitive way of achieving this goal.

The approach of having an identical copy of a real, physical system - effectively a twin - has been around for quite some time. Famously being used since the 1960s by NASA to verify new procedures from the ground for spacecraft already in orbit, this technique gained momentum in the following years in many other industries as Grieves and Vickers noted in [1]. Because of advancements in processing power it became unnecessary for that twin to be a physical instance itself. Starting with CAD models, where there is no direct feedback from the product to the now digital twin, evolving all the way to DTs which are constantly fed by data from their real world counterparts [1]. A recent example is an Earth observation DT currently in development. It is supplied with data from both space and ground based sensors to achieve better weather forecasting [2].

The formal beginning of DTs, albeit as a tool for product life-cycle management, happened at a University of Michigan presentation in 2002 by Dr. Grieves. Since then it has become a widely used technique to monitor, test and experiment with real world systems in a controlled and digital environment [1].

Both Wu et al. [3] and Vaezi et al. [4] have noted that despite promising use cases, DTs in computer networking have not gained as much traction - at least for the time being. One such use case is the simulation of new routing strategies via a DT and the subsequent ability to predict the real networking behavior. Those can be applied for example to emulate, validate and optimize 5G network roll-out as Nguyen et al. propose in [5]. Some telecommunication companies already use DTNs for exactly that

 doi: 10.2313/NET-2023-11-1_09

purpose [6] or do see future use cases in implementing next generation mobile networks [7].

3. Definitions and Terminology

While there is broad consensus in the literature about the definition of DT, the term DTN has various meanings. The following section will define the two terms for the scope of this paper and describe them in greater detail.

3.1. Digital Twin (DT)

The optimal DT is a perfect representation of a yet to manufacture or already existing product that holds at least as much information about the real product as the real product itself could. With that information it is possible to create a physical copy of the DT and vice versa. This definition also encompasses the state of the DT when its physical counterpart is already in existence and therefore linked to it for the remainder of the life cycle or beyond. This digital representation can be connected to a single physical entity or to a more complex system comprised of multiple objects. The optimal DT should also hold information about all previous and current states of the physical twin to enable the prediction of its future behavior. [1]

Vaezi et al. summarize slightly varying definitions in the literature down to three distinct entities that are needed: the represented physical system, the DT itself and a communication or information link between the two. [4]

Due to this information-connection DTs have been able to evolve from being simple digital copies to complex representations that change and develop with their respective PS. This link is what enables the DT's broad information capability in the first place. [3] [4]

3.2. Digital Twin Network (DTN)

The term is sometimes used to describe a network consisting of at least two general DTs communicating with each other. However, this paper is about the application of DTs in computer networking specifically. Therefore the term will be utilized in accordance with ITU-T recommendation Y.3090: "*A digital twin network is a virtual representation of a physical network*" [8]. In other words, it is explicitly used to describe the virtual counterpart of a yet to realize or already existing physical computer network. Figure 1 clarifies this definition in a simple example, depicting a PS consisting of a server and three clients that communicate with each other. The DTN is the virtual copy of this physical network with every real hardware and communication link having a virtual counterpart. The bidirectional information connection between the PS and DTN enables the correct representation.

4. Properties and Requirements of DTs and DTNs

There need to be some representable properties to be able to qualitatively and quantitatively describe DTs in detail. Vaezi et al. [4], Minerva et al. [9] and [3] give some examples which are compounded in Table 1 for easier reference.

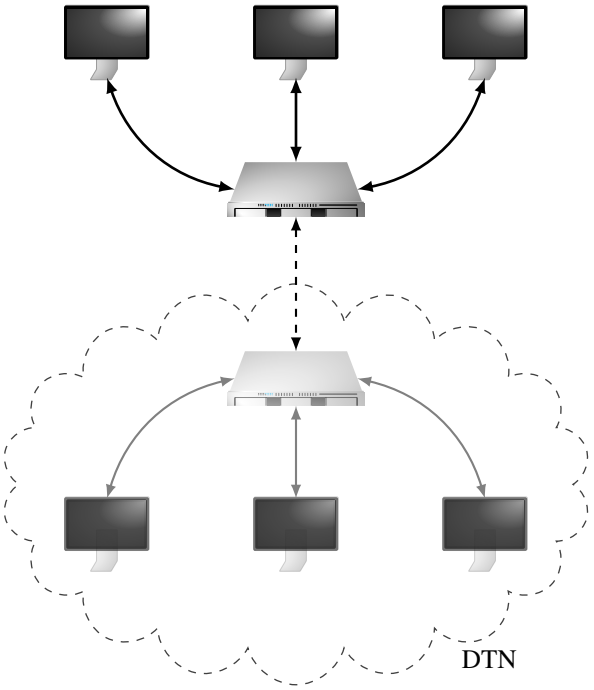


Figure 1: Representation of a physical network and its digital twin network (DTN)

TABLE 1: Properties of DTs based on [3] [4] [9]

Property	**Explanation**
Promptness	Intuitively, this is the reaction time of the DT to its PS. Or more generally the time it takes for information from the PS to reach the DT and change its state.
Similarity	Reflects the difference of information between the PS and DT in the same state. For example, transmission noise can alter the signal and therefore the information.
Replicability	Refers to the ability to replicate a single PS into multiple DTs at the same time. This should also encompass the possibility to create a PS from an existing DT.
Composability	Describes the ability of multiple smaller DTs to be integrated into fewer but larger ones. This adds the capability to form ever more complex systems.
Scalability	If a PS grows in system size, it is desirable for the DT not to increase its network footprint exponentially. Scale in this context also includes geographical size.
Reliability	A DT is considered to be reliable when its data and operational integrity can be verified. This also includes the persistency and availability of information.
Predictability	Since DTs can be comprised of vast amounts of data, they still need to be predictable in precise environments when simulating their behavior with other entities.
Accountability	As a DT can be a set of smaller DTs, the collected information has to be traceable to a specific DT. This property not only includes origin but also the ownership of data and its usage rights.
Adaptability	Is the capability of a DT to adapt in accordance with its PS dynamically. This can involve new models, different data collection or change in needed network resources.

ITU-T recommendation Y.3090 assigns some of the properties mentioned in Table 1 discrete value requirements. For example, the reliability level of a DTN is specified to be at least 99.99%. A selection of requirements can be found in Table 2. The interested reader is kindly referred to [8] for the complete set.

TABLE 2: Requirements for DTNs based on [8]

Requirement	Explanation
Data collection	For a DTN to work, it has to collect vast amounts of data in an efficient manner. The required type of data can vary with use cases but may include: logs, records and status of all network elements; flow statistics like latency, throughput or packet loss; device-specific data such as port information or link status.
Data repository	Storage and retrieval of data is essential to the intended capability of DTNs. Huge amounts of data have to be stored in a way to allow parallel processing and real-time access. For backups and rollbacks it is required to have historical data at hand.
Security	As there is a significant amount of stored information, security is very important. Therefore, the DTN should be able to defend against already happening attacks to the physical network and also attacks against itself.
Privacy	Data protection laws are as applicable to the DTN as they are to the physical network. A DTN must be able to comply with these rules both within its own layer and during communication with the PS.
Compatibility	To fully support future technology, the DTN should be compatible and adhere to established network standards and support various physical interfaces and topologies, different types of databases as well as existing network measurement tools.

Outlined above and in Section 3.1, a DT will and should have comprehensive data from and about its corresponding PS. It has to be noted that in reality the amount of information which a DT can hold is limited. Possible reasons for this are [4]:

- updating of states in regular intervals leads to discrete data points which implies missing information;
- every form of processing by the DT has to take at least some time so delays are inevitable;
- there are limits in terms of resources available to the DT so the capabilities are limited as well.

Nevertheless, a DT is considered to be fully functional when it delivers information with accuracy in an expected and acceptable range while working within these constraints and satisfying the mentioned requirements. It does not have to represent the PS as closely as possible, just as closely as required for the specific use case [4].

5. Modeling a DTN

As of yet, there is no standard model for DTNs so the following example is exactly that: one example. The simple structure in Figure 1 is meant as a quick visual representation. What it lacks to be a real world usable model is a third layer that adds network applications and their requirements for the PS. One such possible description is given in [8] and by Almasan et al. in [10].

This model is comprised of three layers: the physical network is at the bottom, the digital twin network is the middle layer and the top layer is the network applications layer. Figure 2 gives a visual representation for this model.

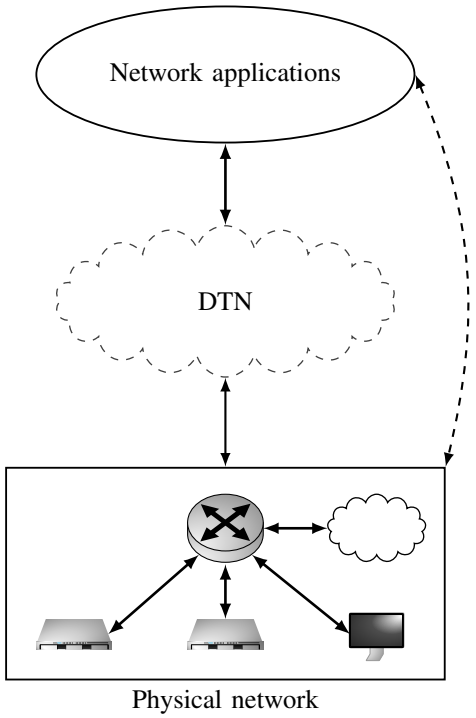


Figure 2: Visualization of the three-layer DTN based on models in [8] and [10]

The leading thought of this model is that instead of deploying network services directly to the PS, they are first fed into the DTN. This way the DTN exposes the capabilities of the physical network to the network applications. In turn, it receives information about the intention of the network application. Based on collected data from the physical network and modeled for example through the use of machine learning on previous traffic, the virtual twin now generates predicted network metrics. These can be used to analyze, verify and optimize the network application services and deploy them to the PS - possibly even without human intervention [8] [10]. This represents the most obvious differentiating factor to a mere simulation since the DTN is able to both control and manage the PS.

Following, the three layers are described in more detail.

5.1. Bottom Layer - Physical Network

This layer is comprised of all physical devices in the network. It is connected to and shares extensive amounts of data with the DTN. The extent of this exchange depends on the specific use case and its requirements. See Sections 3 and 4 for further information. [8]

5.2. Middle layer - DTN

There are three main subsystems that can be identified in a DTN and their interaction attempts to fulfill the requirements given in Table 2. [8]

5.2.1. Unified Data Repository. Responsibilities include the collection and storage of data via the southbound connections to the physical layer but also the process of updating the current state.

5.2.2. Unified Data Models. This subsystem offers, among others, instances of modeled network applications which in turn allow the prediction and data processing capabilities of the DTN.

5.2.3. Digital Twin Entity Management. This allows easy maintenance, logging, control and visualization of the DTN and its data models. It is also responsible for the internal and external security of the DTN.

5.3. Top Layer - Network Applications

The top layer is comprised of the network applications and their services. It is connected to the DTN and relays current or prototype services and requirements to the DTN which then sets up new emulating instances. Upon verification that predefined metrics, such as traffic parameters, coincide with modeled and expected behavior, the DTN proceeds with deploying those services to the physical layer. [8]

6. Practical Applications

There is one empowering capability that the literature is in consensus about: the predictive potential of DTNs [4] [5] [6] [7] [10]. So it seems obvious that this is a centerpiece of many use cases - some of which will be discussed in this section.

- **Planning and construction:** Wang et al. discuss in [11] the possibility of DTNs for use during the complete life-cycle of a network. Because of DTs' property of replicability it is possible to create a new physical system from the DTs' information alone. So it is feasible to begin the implementation of a physical network with the creation of a DTN, continue with running numerous scenarios, optimizing and verifying the network, all before a single physical device is installed. This not only reduces cost but also risk especially with rising system complexity. [10] [11]
- **Maintenance and troubleshooting:** A DTN with its vast amount of meta information about a network has the potential to find the root of an error very quickly. Possible solutions can then be verified inside the DTN and transferred to the physical network. [5] [11]
- **Detection of abnormal behavior:** A functioning DTN mirrors the current operating state of its network which means that deviations from the expected behavior in the physical traffic from the one in the DTN can be telltale signs for anomalies. This can lead to earlier recognition or even prevention of errors. [10]
- **Optimization:** Through the use of various data driven models, a DTN can offer easier network optimization capability because it does not impact the current active physical network. One particular category of models can offer very promising outcomes: machine learning technologies. They can benefit from faster and more efficient operation compared to traditional optimization algorithms because of their awareness of previous optimizing runs. [8] [10] [11]
- **Innovation:** Trial runs in physical networks can have multitudes of negative outcomes due to various reasons for example unavailability of the network during the run or risk of damage to the system. Naturally, network operators are therefore very cautious and conservative when testing innovative technology in real world surroundings. A DTN offers an environment in which new protocols, applications or devices can be scrutinized in a safe manner. It also provides the possibility for in-advance-testing of edge cases like network failures, misconfigurations or security breaches. [8] [10]

7. Discussion and Open Questions

Condensing the findings of this paper, DTNs offer many capabilities and advantages, especially for complex and big networks. Because of the deep interaction with the PS, the DTN is able to manage and control the physical network in real-time.

What remains to be discussed however, is the practical implementation of a DTN that fully satisfies the presented requirements and delivers on the promises. Since tech companies that already use DTNs are generally anything but open about their implementations, this is a big obstacle in experimenting and iterating over different design possibilities of DTNs. Generalizing complex and huge network topologies while still allowing for those predictive capabilities is an open research question. Furthermore, the collection, storage and processing of the desired amount of data is everything but trivial, especially in real time environments. [3] [10]

Future work is necessary to come up with possible implementations that do not rely on broad financial and subsequent technical capabilities which currently only big tech companies seem to be able to provide. Further research into DTN technology for small and medium sized networks can also shed light on whether those would benefit from this technology as well.

8. Conclusion

In this paper the technology of DTs was introduced and their application in computer networking was discussed. These DTNs can offer solutions to questions concerning vast and interconnected networks like mobile telecommunication networks. Properties of and requirements for DTs and DTNs from different sources of literature [3] [4] [8] [9] were identified and presented, as well as the suggestion for a conceptual model of a DTN [8] [10]. Applicable use cases include the complete life cycle management of a network spanning from planning all the way to maintenance, troubleshooting and upgrading [5] [11]. The predictive capabilities of DTNs also allow for the detection of abnormal network behavior and time and cost effective integration of new technologies into the network [10]. One of the most promising areas of interest is the optimization of network traffic and topology. With

its rich amount of data about the state of a network, the DTN technology in combination with machine learning algorithms can offer faster and probably better optimization results than current network simulation [10] [11]. Lastly the question of actual implementation was discussed and underlying problems presented like the requirement for storing and processing huge amounts of data in real-time. In case these problems are successfully tackled, DTNs have many possible applications for designing, optimizing and maintaining computer networks.

References

[1] M. Grieves and J. Vickers, *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. Springer International Publishing, 2017, pp. 85–113.

[2] S. Chin, "Using the Digital Twin to Improve Weather Forecasting," https://www.designnews.com/artificial-intelligence/using-digital-twin-improve-weather-forecasting, 2022, [Online; accessed 20-March-2023].

[3] Y. Wu, K. Zhang, and Y. Zhang, "Digital Twin Networks: A Survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 789–13 804, 2021.

[4] M. Vaezi, K. Noroozi, T. D. Todd, D. Zhao, G. Karakostas, H. Wu, and X. Shen, "Digital Twins From a Networking Perspective," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 525–23 544, 2022.

[5] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital Twin for 5G and Beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, 2021.

[6] S. Communications, "Simplifying 5G with a Network Digital Twin," https://www.spirent.com/assets/wp_simplifying-5g-with-the-network-digital-twin, Tech. Rep., 2022, [Online; accessed 30-March-2023].

[7] P. Öhlén, "The Future of Digital Twins: What will they mean for Mobile Networks?" https://www.ericsson.com/en/blog/2021/7/future-digital-twins-in-mobile-networks, 2021, [Online; accessed 30-March-2023].

[8] "ITU-T Y.3090; Digital Twin Network – Requirements and Architecture," http://handle.itu.int/11.1002/1000/14852, International Telecommunication Union, Recommendation, Feb. 2022.

[9] R. Minerva, G. M. Lee, and N. Crespi, "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.

[10] P. Almasan, M. F. Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. R. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Digital Twin Network: Opportunities and Challenges," *CoRR*, vol. abs/2201.01144, 2022. [Online]. Available: https://arxiv.org/abs/2201.01144

[11] D. Wang, J. Guo, Y. Ouyang, S. Wang, A. Yang, Z. Ren, Y. Ding, G. Chen, C. Zhou, and D. Chen, "Leverage Digital Twins Technology for Network Lifecycle Management," in *2022 IEEE 2nd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2022, pp. 1–5.

Positioning in 5G Networks - Overview and Security Threats

Lukas Wittmer, Leander Seidlitz, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: lukas.wittmer@tum.de, seidlitz@net.in.tum.de, andre@net.in.tum.de

***Abstract*—Determining your position is something humans always wanted to do. In present times many different technologies, like GPS, assist this process. As Satellite-based positioning struggles in environments where the sky is obstructed, like forests or indoors, new technology has to cover these areas. This technology is positioning via the 5G mobile communications network as the 5G coverage grows better and better. This paper will provide an overview of positioning metrics, like Time-of-Arrival measurements, and the approaches to determine an exact location from these measurements. Furthermore this paper gives an overview of the security aspects of the 5G positioning ecosystem. This overview includes possible attack targets, like the Location-Information-Service-Provider, and possible threats to the system and their consequences. These threats range from interferences to Man-in-the-Middle-Attacks by an active attacker, which can steal or alter the location information of a user.**

***Index Terms*—5G networks, positioning, security**

1. Introduction

With 5G rising in popularity and its coverage growing dense, other features than communication are becoming much more viable. One of these features is the increased usability for positioning a device located in the 5G network. This is especially useful in indoor environments where traditional positioning techniques like the Global Navigation Satellite System (GNSS), better known as one of its subsystems GPS, fail to provide sufficient coverage. This paper aims to explain the basics of 5G-based positioning by covering the most relevant metrics and approaches to archieve an accurate position estimate. Furthermore, as security is and always will be a relevant topic in mobile communications, this paper will explore the different threats and attack targets a 5G positioning system offers.

1.1. Related Work

With 5G positioning being a topic of current research, an increasing number of papers have been published dealing with different aspects of position estimation using the 5G infrastructure. Some of these papers deal with similar topics to this paper and are referenced in the following section.

A paper that is similar to this paper is "Positioning in 5G and 6G networks—A Survey" by Mogyorósi, Revisnyei et al. [1]. It provides an overview of positioning methods while focusing on machine learning assisted approaches. It also provides an outlook on the influence of the introduction of 6G on the presented approaches.

Another paper with a similar topic is "A Look at the Recent Wireless Positioning Techniques With a Focus on Algorithms for Moving Receivers" by Tahat, Kaddoum et al. [2]. This paper deals with the general use of wireless networks for positioning. The wireless networks that are referenced in this paper include Bluetooth, WLAN and RFID.

The book " A comprehensive guide to 5G security" by Liyanage, Madhusanka et al. [3] deals with the security aspect of 5G positioning. After giving an overview of positioning mechanisms in 5G, it deals with threats to security in the 5G positioning network. It also provides an overview of the security mechanisms of 5G that help to mitigate the mentioned threats.

2. Measurement Modes

Positioning in mobile environments can be done in different ways. These ways differ in the active communicators, like the base stations, the device whose position is to be estimated and possible other devices helping with the positioning. They also differ in the devices calculating the distance.

The first measurement mode to be presented, is network-centric positioning. In network-centric positioning the network is doing the computation, while the device to be located is sending signals needed for measuring and receiving its estimated position. For the user device to use the estimated position in e. g. a navigation application, the position data has to be sent to the device via the network. A graphic representation of this approach can be seen in Figure 1 (a).

In device-centric positioning, the roles of the network and the device are reversed. While the network sends measuring signals, the device receives them and does the position computation on its own. Therefore, no position estimates need to be sent over the network, which is beneficial to privacy (more on that in Section 5). An example of a device-centric architecture can be seen in Figure 1 (b).

While device- and network-centric positioning are two opposed approaches, both can be supplemented by cooperative positioning. In Figure 1 (c) one can see that while doing cooperative positioning, other devices, that are not base-stations of the network can also send measurement signals or their position estimates to the device to be

 doi: 10.2313/NET-2023-11-1_10

located. This additional data increases the accuracy of the estimated position.

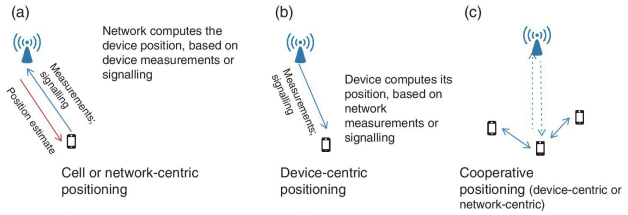


Figure 1: Different measurement modes [3]

In the rest of this paper network centric positioning will be used as the default measurement mode, but the metrics and approaches presented, work with other modes. The explanations in this chapter are based on [3, Chapter 13].

3. Position Metrics

The metric that takes the least computational effort that can be used to estimate the position of a device is whether its present within a certain base station's range. This metric is known as Cell-ID and can be used to give a rough position estimation of the device as described by Larsson in [4].

A metric for accurate positioning is the Time of Arrival (ToA). When the sending time of a signal is known, the time in flight of the signal can be calculated. From this, the distance between the origin of the signal to the receiver can be computed. This metric needs the sender and receiver to sync their clocks, in order to provide accurate measurements. The modus operandi for such a measurement is described by Van Haute, Verbeke et al. in [5].

When synchronisation of receiver and sender is not feasible, ToA measurements are not possible. If the distance measurement should still use time as its primary measurement, the time difference between the arrival of multiple signals, known as Time Difference of Arrival (TDoA), can be used. In this approach, the receivers still have to be synchronised, but no synchronization between the sender and receiver is necessary. A brief overview of TDoA measurement is given by Tahat et al. in [2].

Another metric for positioning is the Angle of Arrival (AoA) or Direction of Arrival (DoA). This metric uses the fact that base stations receive the same signal on multiple antennas. The angles at which the signal arrives at the receiver can be computed by measuring the time difference with which it arrives at the different antennas of the receiver. When this measurement is combined with the knowledge of the distance between the antennas at the receiver, the angle of Arrival can be computed, using trigonometric functions. A more in-depth analysis of this metric is provided by Tuncer and Friedlander in [6]. This metric is supported by 5G in particular, as 5G base-stations are equipped with up to thousands of antennas, providing a good basis for AoA measurements [1].

The Received Signal Strength (RSS) is one of the oldest metrics used in positioning. This metric dates back to 1969, as it was developed as a method to locate moving vehicles, e. g. police cars as stated by Figel, Sheperd and Trammell in [7]. This metric uses the fact that the signal strength scales with the distance it has to traverse. The method to derive a distance from the strength of the received signal is described by So and Lin in [8]. A problem with RSS as a metric is that it is susceptible to environment interference, as described in [2].

4. Approaches to Positioning

As the different metrics explained in the previous chapter, except for the Cell-ID approach, compute only a distance or an angle but no location, these measurements need to be processed further. This processing can happen differently based on the chosen approach.

4.1. Cell-ID-based Positioning

Cell-ID-based positioning is a basic form of positioning in mobile networks. In this approach the location of the connected base station is assumed as its location, as described by del Peral-Rosado, Raulefs et al. in [9]. However, this technique is imprecise in regions with low population density and sparse cell coverage as a single base station covers a large area in these regions. Better accuracy can be archieved by this method, if multiple base-stations are combined [1]. If a device is or was recently present in more than one base-stations cell the area in which the device is probably located can be narrowed down to the area that is covered by both base-stations.

4.2. Angle-based Positioning

Angle-based positioning is one of two approaches that use geometric properties. It uses the AoA metric to determine the position of a device. With the position of the base station and the angle at which the signal arrived, a Line of Bearing (LOB) can be computed. If the LOBs of multiple base stations are combined, the device can be positioned at the intersection of the lines. As the intersection of lines only requires two LOBs, two base stations are sufficient for an angle-based positioning approach [2]. More base-stations can help to improve the accuracy by eliminating outliers or false measurements. This approach is also known as triangulation. A challenge to the angle based approaches are the non line of sight (NLOS) conditions. The NLOS conditions are that without a line of sight between the device and a base station the accuracy of the corresponding measurement drops dramatically, which can lead to a wrong position estimation due to multipath propagation of the signal. Multipath propagation describes the reflection of a radio signal from surfaces and can therefore existence of multiple paths from the sender to the receiver. This results in different angles from which the signal can arrive. Approaches using ToA or TDoA (see Section 4.3 and 4.4) also struggle with NLOS conditions but not to the extent of angle based approaches, as there are methods to mitigate their effects in timing based systems.

4.3. Range-based Positioning

Another approach that uses geometric properties is range-based positioning. This approach uses the distance

 doi: 10.2313/NET-2023-11-1_10

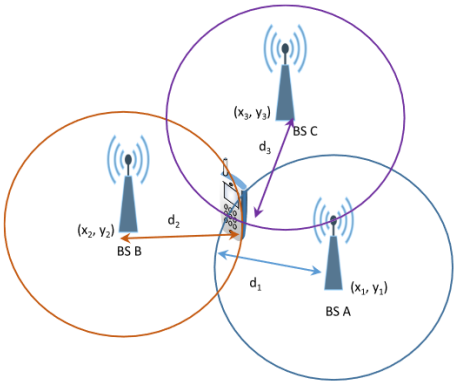


Figure 2: Range-based positioning with RSS or ToA [2]

from the device to multiple base stations. This distance can be derived from time measurements (ToA or TDoA metric) or signal strength (RSS metric). Range-based positioning needs at least three involved base-stations, instead of the two needed by angle-based positioning. While the geometric representation for the position estimation involving RSS and ToA is using the intersection of three circles around the base stations, with the respective distances as radii, also known as trilateration (as seen in Figure 2) the TDoA approach instead uses the intersection of hyperbolas around the base stations as no ranges but only range differences are know [2].

4.4. Fingerprinting-based Positioning

While the previously mentioned approaches use ad hoc measurements, fingerprinting-based positioning uses another method to estimate positions. The typical procedure for this approach consists of two phases: the offline or training phase and the online phase. During the training phase, the observed area is divided into a grid of measurement points. Multiple measurements of the chosen metric are then acquired at each measurement point. These metrics can be any metric from Cell ID to RSS, but some, like RSS, are more suitable to be used by a fingerprinting approach. These metrics are combined in a fingerprint that is identifying each measurement point. These fingerprints are inserted into a database that is used during the online phase. In the online phase the same measurements are repeated and combined into another fingerprint. This fingerprint is matched against the fingerprints within the database generated in the offline phase. As these measurements are influenced by NLOS conditions (Section) 4.2), the measurements are extended to minimize the effect of these conditions. As the grid divides the measurement area into discrete intervals, but the position is on a continuous scale, the coordinates of the nearest points are averaged to find the position. This procedure is described using RSS measurements by Yu, Jiang et al. in [10].

4.5. AGNSS Positioning

Assisted Global Navigation Satellite System (AGNSS) positioning is an approach that uses the mobile network differently than others. Its primary use is for indoor positioning, while still using a Global Navigation Satellite System (GNSS) like GPS. As these systems struggle to lock onto their positioning satellites while indoors due to effects like NLOS conditions (Section 4.2), additional data transmitted via a 5G network can help the devices to lock onto the satellites. The functionality of AGNSS is described by Mautz in [11].

5. Security

As a device's location is sensitive information, it attracts the attention of parties that want to abuse it. This abuse can range from location-specific advertisements without the users' permission, to attackers altering the reported location of a device to steal it. To better understand the vulnerability of the 5G positioning system, the attack targets and security threats are shown in the following sections.

5.1. Attack Targets

In the 5G positioning network, every participant can be an attack target, but not every target is equally susceptible to every threat. The three main categories of targets in attacks on 5G positioning are the Location Information Service Provider (LISP), the Location Based Service Provider (LBSP) and the User Equipment (UE). The UE can be divided further into the end-user equipment and the Location Information Collaborators (LIC).

The LISP is the provider of the positioning infrastructure, e. g. the base stations that either send out the measurement signals in device-centric positioning or receive the measurement signals in network-centric positioning. They also provide the LBSP access to their positioning database, if the positioning is network-centric. In a device-centric environment the location is provided by the device itself.

The LBSP, on the other hand, is the provider of the service the UE needs its location for. This service can be a navigation app, a running app or even the tracking of autonomous robots in, e.g. a factory.

The end-user device is by far the most commonly known participant in the system. This device can be any device with access to the 5G network, like a mobile phone, a car with a built-in SIM-Card or a mobile robot in a factory.

The LIC is an optional participant in the system as its only present in a cooperative positioning environment. LICs can be from the same device spectrum as the end-user device and help the end-user device estimating its position, as 5G allows device-to-device communication. While using an LIC can improve the accuracy of position estimation, it is also the device type with the greatest potential to be malicious.

Figure 3 shows important traffic that for position estimation, like position or service data, is transported via the UE. This fact makes the UE a prime target for attacks on location privacy.

5.2. Security Threats

The threats to security in the 5G positioning system can be categorized by the participants they affect. These

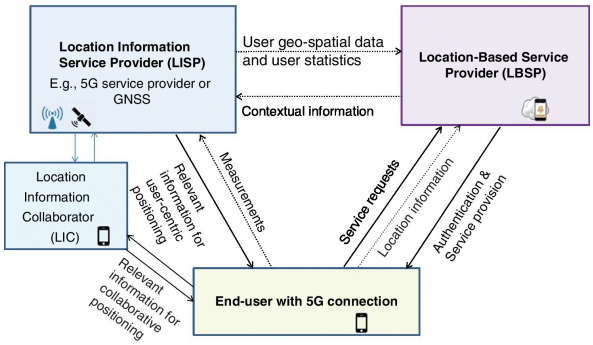


Figure 3: 5G positioning targets [3]

threats can have different effects on the system, ranging from a financial loss at an LBSP, e. g. if a user can use the service without paying for it, up to position information theft, e. g. if a location database gets compromised.

The first category of threats are threats affecting multiple participants or every participant of the system. This category includes Denial-of-Service- or Distributed-Denial-of-service-attacks that saturate any participant, therefore making it unavailable for the whole system. Depending on the attacked participant, this can either decrease the estimations accuracy, e. g. if only a single base station is unavailable, or render the whole system inoperable, e. g. if the UE is saturated. Also in this category is the Man-in-the-Middle (MiM)-attack. In this context, the main target for a MiM-attack would be the communication between the UE and the LBSP, as this communication will include information about the location and authentication. An eavesdropping attack is also possible in a network-centric system. If an attacker has multiple nodes positioned in the area covered by the target UE it can record the measurement signals emitted by the UE and perform a TDoA, AoA or RSS measurement by itself. A ToA measurement is impossible in this scenario as the attacker is purely passive and therefore can not synchronize its clocks with the UE.

The next category includes threats that affect the LISP. One of these threats is the presence of malicious nodes in the system that send fake signals to the base stations, to create errors in the position estimate. Another threat to the LISP, that does not require an attacker, is interference. As 5G is using radio signals to send messages, these signals can be jammed either intentionally by an attacker, or unintentionally by other participants or natural causes, like a storm. The jamming of signals can alter or prevent measurements, making the positioning system unusable.

Threats that affect the LBSP are also a category of threats that the 5G positioning system has to deal with. This category includes the unauthorized use of the LBSPs service, either being unpaid use, in case of a paid service, by hijacking the signals or the misuse of the service, e.g. an employer tracking the phones of his employees to detect if they are working or not. Another threat to the LBSP is the leakage or theft of information from the LBSP database. This threat is a concern for privacy reasons, as the location of any user using an LBSPs service e. g. Google Maps, could be obtained if the location database is breached.

As stated in the closing paragraph of Section 5.1, the UE may be the prime target for an attack as it is the central part of the positioning calculation. It also is the weakest part of the system, from a security perspective, as the users of the UE tend to be careless about restricting how their data is used, as described by Gašparović, Nicolau and Marques in [12]. A threat that comes from this category is location-tracking malware that is installed directly on the UE. This malware can report the location of the UE without the user knowing. Another threat to the UE is the loss of accuracy when using a fingerprinting approach that can come from errors in the communication between the measuring UE and the LISP while building the database in the training phase. These errors, if not corrected, can lead to inaccuracies in the position estimation during the online phase of the positioning. Another threat for the UE, that is also a threat to the LISP, is interference, as they prevent both the LISP and the UE from communicating. Another threat that was mentioned above is location theft. When location theft is mentioned in the context of the UE, it is not about the location of a device becoming public but rather about a device reporting a fake location. This can result in e. g. identity theft if a user is authenticated via his location. A deeper explanation of these threats and others, as well as some methods for protection against them, can be found in [3, Chapter 13].

6. Conclusion and future work

We gave an overview of the current positioning metrics and approaches using the 5G mobile communications network. Additionally, we presented an overview of attack targets and threats to security in the 5G positioning ecosystem. The 5G system provides several metrics for position estimation, including Time-of-Arrival, Angle-of-Arrival and Received-Signal-Strength. Multiple measurements of these metrics have to be combined to result in an exact position. The approach and the metrics that are used for estimating the position determine the minimum of required base stations to determine an exact location. Range-based approaches, that use trilateration as their theoretical foundation, need three base-stations while angle based approaches, using the AoA metric and triangulation as their theoretical foundation, need only two base stations. Additional base stations can increase the accuracy by providing additional information or by elimination outliers, caused by interferences or NLOS conditions. From the security point of view, the system provides three main attack targets. The LISP, the LBSP and the UE, are susceptible to several threats, that are either unintentional, like interferences caused by storms, or intentional, like DoS-Attacks or MiM-Attacks by an adversary.

As the newer generation of mobile communications is rising on the horizon with 6G, it will be interesting to see what additional features for positioning come with it and how current metrics and approaches will be improved. Furthermore, it will be interesting to dive deeper into the security topic and analyse different concrete attacks on the mobile positioning architecture. Methods to prevent these attacks could be learned and the security of the infrastructure and location privacy could be improved. These two topics provide a first basis for possible future research and papers.

References

[1] F. Mogyorósi, P. Revisnyei, A. Pašić, Z. Papp, I. Törös, P. Varga, and A. Pašić, "Positioning in 5g and 6g networks—a survey," *Sensors*, vol. 22, no. 13, p. 4757, 2022.

[2] A. Tahat, G. Kaddoum, S. Yousefi, S. Valaee, and F. Gagnon, "A look at the recent wireless positioning techniques with a focus on algorithms for moving receivers," *IEEE Access*, vol. 4, pp. 6652–6680, 2016.

[3] M. Liyanage, I. Ahmad, A. B. Abro, A. Gurtov, and M. Ylianttila, *A comprehensive guide to 5G security*. Wiley Online Library, 2018.

[4] J. Larsson, "Distance estimation and positioning based on bluetooth low energy technology," 2015.

[5] T. Van Haute, B. Verbeke, E. De Poorter, and I. Moerman, "Optimizing time-of-arrival localization solutions for challenging industrial environments," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1430–1439, 2016.

[6] E. Tuncer and B. Friedlander, *Classical and modern direction-of-arrival estimation*. Academic Press, 2009.

[7] W. Figel, N. Shepherd, and W. Trammell, "Vehicle location by a signal attenuation method," *IEEE Transactions on Vehicular Technology*, vol. 18, no. 3, pp. 105–109, 1969.

[8] H. C. So and L. Lin, "Linear least squares approach for accurate received signal strength based source localization," *IEEE Transactions on Signal Processing*, vol. 59, no. 8, pp. 4035–4040, 2011.

[9] J. A. del Peral-Rosado, R. Raulefs, J. A. López-Salcedo, and G. Seco-Granados, "Survey of cellular mobile radio localization methods: From 1g to 5g," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1124–1148, 2017.

[10] F. Yu, M. Jiang, J. Liang, X. Qin, M. Hu, T. Peng, and X. Hu, "Expansion rss-based indoor localization using 5g wifi signal," in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014, pp. 510–514.

[11] R. Mautz, "Overview of current indoor positioning systems," *Geodezija ir kartografija*, vol. 35, no. 1, pp. 18–22, 2009.

[12] M. Gašparović, P. Nicolau, A. Marques, C. Silva, and L. Marcelino, "On privacy in user tracking mobile applications," in *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2016, pp. 1–6.

 doi: 10.2313/NET-2023-11-1_10

Content and API Acceleration Using Content Delivery Networks

Tom Maximilian von Allwörden, Markus Sosnowski*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: tom.von-allwoerden@tum.de, sosnowski@net.in.tum.de

***Abstract*—Modern web services are accessed all over the world by potentially many users. Content Delivery Networks (CDNs) play an integral role in lowering the latency for end-users that are using these services by caching objects like a website's source code and images close to the user. Traditionally, CDNs have been used for static content, but with the rise of personalized user experiences and dynamically changing content, new requirements for CDNs emerged. APIs are the most common way such content is accessed, so an effort is made to accelerate these.**

This paper will give a brief introduction into how CDNs work, how virtual network optimize routing and then examine how dynamic content and APIs can benefit from CDNs via different invalidation techniques and what tradeoffs to consider. Furthermore the paper collects diverse approaches in the field of edge computing for moving the application logic itself close to end users.

***Index Terms*—cdn, internet, api acceleration, dynamic content, virtual network, edge computing**

1. Introduction

Content Delivery Networks (CDNs) bring web-services like websites or APIs closer to the end-user by deploying various globally distributed replica- or edge servers that cache the content of the origin server. Such locations are called Point of Presences (PoPs) and typically consist of Internet Exchange Points or locations inside of an Internet Service Provider (ISP)'s networks [1]. Sitaraman et al. [2] show the main business incentive: high load times of a website can avert potential users away from a commercial site. A large CDN provider like Akamai therefore deploys over 170 000 edge servers in about 1300 networks to accommodate todays needs [3,4], especially in times where latency is the single biggest influence on web performance, according to Ilya Grigorik [5].

When a user tries to access a web-service, instead of visiting the origin server directly, they get directed to a geographically close edge server by the CDN's request-routing system. If the content is cached, it is then immediately returned, thus lowering the latency; however if the content is not cached the CDN must fetch it from the origin [6]. This can be done effectively using a CDN's virtual overlay network [2].

Beheshti [7] mentions that caching is especially effective for static content like images or script files, but dynamic content and APIs are traditionally harder to cache because simple caching could lead to the user getting stale, outdated content.

This paper is structured as follows: Section 3 presents an overview of different request routing methods. We take a closer look at before mentioned virtual networks and how connections to the origin are accelerated in Section 4. Section 5 is about handling dynamic content with a focus on a technique called invalidation. Lastly, in the umbrella edge computing fall various approaches that improve a site's performance, which we discuss in Section 6.

2. Related Work

Two of the focal points of this paper are cache invalidation and consistency. We will not touch on the protocols that different CDNs use to implement their invalidation strategies, such as leases. This is discussed in more detail by Ninan et al. in [8]. Wingerath et al. describe their approach for handling dynamic content in [9]. They focus on the refresh procedure to detect stale content and afterwards invalidate it automatically. We, on the other hand, will look at manual invalidation, where we invalidate instantly when events cause data to become stale.

3. Request Routing

Katz-Bassett et al. [10] state that the request-routing system is responsible for directing a user from the origin server to one of the CDN edge servers. There are three common approaches CDNs use: Domain Name System (DNS)-based, IP anycast-based and HTTP 304-based redirecting, where the user first contacts the origin, which then redirects the user to a CDN server. We take a look at the former two in more detail.

Gang Peng [6] presents more advanced methods, such as those based on Peer-2-Peer systems, but these are beyond the scope of this paper.

3.1. Domain Name System

With the DNS-based approach, which is covered by Hung et al. [11], the administrator of a domain origin places a CNAME entry pointing to the CDN's DNS server entry in its domain name server.

When a user requests a website, their local DNS (LDNS) resolver, most commonly provided by their ISP, will recursively query the domain name and contact the CDN's DNS server. This server then selects the edge server PoP based on the LDNS IP-address [11]. This is usually a good indicator for the approximate geographical location of a user when not using a public resolver [10].

 doi: 10.2313/NET-2023-11-1_11

Other metrics, such as the network load and previous edge server choices are also taken into account when selecting the edge server within a PoP [2].

Notice that with this scheme the end user's IP-address is not directly used for selecting an edge server. Sitaraman et al. [12] and [10] discuss the proposed EDNS-Client-Subnet DNS extension that allows the LDNS server to send the user's /24 IP-address-prefix along with the request, thus allowing the CDN to make a better decision.

This approach is widely popular. Examples of CDNs using DNS-based approaches include Akamai [2], Edgenext [13] and AWS Cloudfront [14].

3.2. IP Anycast

With the IP Anycast approach, every edge server is assigned the same IP-address. This method makes use of the fact that when a network router gets the same route announced from different interfaces, it chooses only the one with the lowest hop count. A disadvantage with this is that CDNs have less control over which edge server the client connects to, but studies have shown that the method leads to the optimal edge server in 80% of the time [10].

Popular CDNs, which use this approach include Cloudflare [10], Microsoft Azure Front Door [15] and Google Cloud CDN [16].

4. Virtual Network

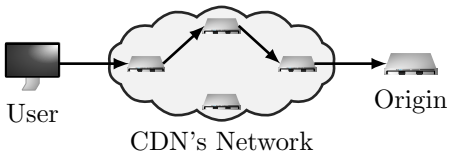


Figure 1: Routing using a CDN's virtual network

Classical internet routing has several disadvantages: The Border-Gateway Protocol that ISPs use to exchange routes is known to lead to sub-optimal routing since it has no topology information and solely uses hop-count as its main metric [2]. The propagation of failed routes can also take a significant amount of time. Furthermore, ISPs are also driven by financial incentives and could prefer routes over cheaper but slower peering-partner compared to costly Customer-Provider (C2P) connections [2].

To mitigate this, CDNs influence the routing path of a package by adding their servers as intermediate hops, resulting in an overall more optimal route. They do this by continuously measuring latency and package drop between their edge servers and taking the topology information of their server locations into account [2].

Some big players like AWS Cloudfront [17] and Microsoft Azure [18] come with their own backbone infrastructure between different PoPs, circumventing suboptimal internet routing even further. Common optimizations when using the virtual network include:

Long-Term TCP& SSL connections A user's TCP & SSL connections are terminated close to them at the edge. From there on, the edge server uses pre-established TCP & SSL connections between the origin and various edge servers. As the respective handshakes require several round-trips, the saved cost quickly accumulates [2,19].

No slow start TCP normally begins in a "slow start" phase with small window sizes. CDNs skip this phase and choose larger window sizes by taking the previously mentioned network measurements into account [2,19].

Data Compression : Images and other objects can be compressed on the edge close to the origin server, and thereby loaded quicker [2,19].

Abundant packages : CDNs can try to send the same packages over multiple routes and then take the one that arrives first. This also helps against package loss, but can cause more congestion [2].

Virtual networks thus accelerate the content fetch from the origin in case of a cache-miss and are therefore important for uncacheable and dynamic content. It was shown, that in Asia the use of a CDN's more optimized routing can lead to a 30-50% performance improvement [2].

5. Caching of Dynamic Content

In the last section we saw a general way to speed up requests to the origin, but we did not make use of a CDN's caching capability yet. This section introduces dynamic content and discuss techniques of caching it.

Dynamic content is content that changes over time but might be the same for all users while personalized content refers to dynamic content that is different for each user. For example, a site presenting the top 10 most viewed news articles of the day is dynamic content, while a user's shopping cart is personalized content. As the storage requirements for the latter scale with the number of users with each entry only relevant for a single user, caching it might be of little value for a service, but we will see a way of handling this content in Section 6.

Lawrence et al. [20] highlights that not all dynamic content is the same: some content is valid for a long time, while other might change with every request made. The update trigger might be an external event, such as a new blog post release, or it might simply be after a certain period of time has passed. The latter would be easier to handle, as one could just choose this period as the cache's Time-To-Live (TTL).

The main problem we face with caching is consistency, as we have multiple, distributed copies of an object that might become stale at any time. But even for dynamic content that normally changes with every request, a site owner might choose to intentionally give up on consistency and decide to cache it for a brief period, a practice referred to as micro-caching, as this alone can take a considerable load off the origin [9]. This is especially true for large services that deal with hundreds to thousands of requests per second.

Another way CDNs can benefit APIs is by accumulating multiple requests over a short period of time and sending them bundled to the origin. Therefore, taking load and overhead off of the origin [21].

As APIs usally interface with a database at the origin, there are efforts that try to map the cache objects to this underlying relational data. The analysis is typically done by observing the traffic between web listeners, databases

and web services, but the analysis of this mapping is too expensive and unreliable to use in practice [20].

Therefore, we will focus on the most common way dynamic content is cached, namely, fine-grained cache control and invalidation.

5.1. Cache Control and Invalidation

In comparison to the difficult and expensive task of maintaining and modeling data dependencies between cache objects, invalidation is a cheap and simple alternative [20]. We want to cache dynamic content for as long as possible and invalidate it as soon as it becomes stale. The most common way invalidation occurs is simply when the set livetime of a cached item expires. This livetime can be controlled via the "Expires" header and is passed in the response from the origin [22]. Another option is to invalidate the cash manually. This can be done via the CDN's API, also refered to as *purging* [23,24].

We generally want to invalidate as little as necessary so that we reduce the number of cache misses. Fine-grained invalidation is generally more difficult to do and costs the CDN more resources. CDNs commonly provide different purging methods that vary in their level of granularity [23,24]:

By URL Invalidates the cache object that is associated with the URL.

By Prefix Invalidate all objects with a given URL prefix

By Tag The origin Server can associate cache objects with a tag by adding a "Cache-Tag" header to the response. This way many possibly by path unrelated endpoints can be purged with a single request.

By Geo Only invalidate cache objects at certain PoPs.

Even with purging, users could still get stale content if the CDN only provides weak consistency. Weak consistency means that the cached objects on the different PoPs only *eventually* get invalidated instead of *instantly* as it is the case with strong consistency [25]. Even CDNs such as Fastly that advertise a "instant purge" feature do not have strong consistency across different PoPs [26]. Although this can already be enough for services, where users requesting the same content tend to be geographically close together. When we look at a typical restful API, there are generally two types of requests [24]:

state requesting GET
state changing POST, PUT, DELETE

The former can usually be cached, while the latter are commonly passed through to the origin [24]. In response to a state changing request, the corresponding cache entries need to be invalidated. Let us explore how invalidation mechanics can be used in practice by examining the following example of a video watching & commenting platform with the following endpoints:

- `PUT /api/<videoid>/comment`: users can leave comments with their username attached.
- `GET & PUT /api/<userid>/profile`: users can retrieve or change their profiles e.g. their username.

In the event that a user changes their username, we want to invalidate all comments made by them to reflect this change. In this scenario, invalidating every object separately is a costly task. Instead, we use tags to associate each comment and possibly other user-related endpoints with the user's id and purge them with one purge call [24,27,28]. Another possibility is to handle purging and other logic on the edge servers themselves. We highlight this in more detail in Section 6.

Further techniques to to improve invalidation include:

Cache-keys The index into the cache to associate an URL with a cached object is called the cache-key. Modern CDNs allow one to include or exclude certain parts and parameters of the URL and request headers into the key to avoid unnecessary cache-misses [29].

Auto revalidation A CDN might try to fetch the newest state of an object once the TTL has expired. If the content was invalidated via a message from the origin, the origin could pass the newest version of that object along [25].

Invalidation order Monitoring the popularity of objects allows a CDN to invalidate the popular ones first as these affect the most users [20].

In recent years, GraphQL APIs have become a modern alternative to RESTful APIs. Wundergraph and Hygraph are examples of services that specialize in GraphQL API management and include CDN-like caching capabilities. These can handle invalidation automatically to some extent, but use relatively simple approaches: Wundergraph allows the developer to define dependencies between APIs but not the data dependencies within an API [5,30]. It caches all objects for 10 seconds. Hygraphs supports invalidation and does this based on the GraphQL schema and content changes, but only with weak consistency [31].

6. Edge Computing

Invalidation is not perfect for applications with a high number of uncacheable requests or whenever we have a cache-miss, as the origin is the sole producer of fresh content and needs to be contacted.

The idea behind edge computing is to move as much application logic as possible close to the user onto edge servers. When a request can be completely handled on the edge, no additional overhead is needed to reach out to the origin server, thus decreasing latency and taking load of the origin. For this, applications are usually split into an edge and an origin component [21].

One popular approach is the deployment of *edge functions*. Edge functions run in response to incomming requests. These can then modify the request, interact with the CDN's caching system, or implement entire API-endpoints on edge, and thus behave similarly to an API-gateway [2]. For example, the invalidation logic discussed in the previous section can be implemented as edge functions. Most CDNs offer such functions, for example AWS Lambda@Edge [32] or Cloudflare Worker [33].

Applications that only need a static database to function are prime candidates to be moved completely onto the edge. For example encyclopedias, dictionaries or product catalogues of e-commerce businesses with a fixed number of products [21]. In the following sections, we will discuss various techniques in more detail.

6.1. Normalization

URL normalization is applied to the HTTP path and parameters of incoming requests and ensures the same

 doi: 10.2313/NET-2023-11-1_11

encoding for all paths in compliance with RFC3986 [34]. As the path is a common component of the cache key, different encodings of the same object can lead to unnecessary cache misses.

For example, these two URLs refer to the same object: "example.com/api/./user/青 沼" and "example.com/api/user/%E9%9D%92%E6%B2%BC". Since these would hit in different cache keys, normalization is applied to map the former path into the latter.

6.2. Edge Site Includes

Personalized sites are created explicitly for the user requesting them, and it thus provides little value for a CDN to cache the site as is, especially because the presented content might become stale fast anyway [20]. But looking at how personalized sites are constructed one might notice that they are often built using shared fragments [20]. For example, Figure 2 shows different fragments of the bing.com website with fragments such as recent news or weather that could be shared across users of the same city.

Edge-Site-Includes (ESI) allow for the creation of dynamic, fragmented websites by assembling the website on the edge. Here, a website consists of an HTTP template with special ESI tags, that describe the type and location of the fragments that should be included in the final site. The fragments are shared between different templates and can each be individually cached and have their own TTL [2,20]. The ESI environment allows one to encapsulate personalized information into a fragment that itself can be cached [20]. A edge server might also speculatively assemble a site beforehand, based on the user's last visited sites [35].

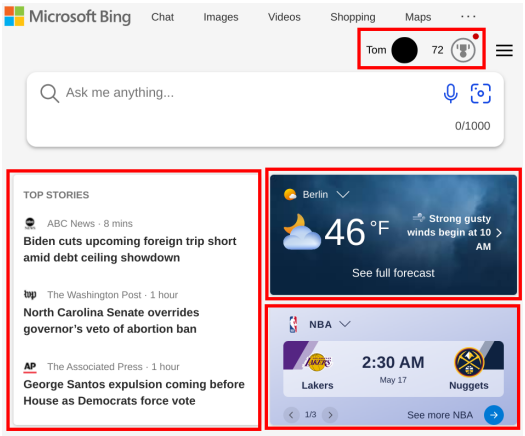


Figure 2: Sample fragmentation of bing.com

6.3. Validation & Authorization

Before forwarding a request to the origin it can be checked for ill-formatted content [2]. A certain kind of validation is authorization, where we check if the user has the necessary access rights. JSON Web Tokens are a popular method to authenticate users. We assume that the user got a API token from an issuer (most likely the origin) in advance. The client then includes this token in the "Authentication" header within each request. The edge which is provided a JSON Web Key Set by the Issuer, checks the user token against this set [36].

Other authorization schemes like OAuth or openid connect can also be handled on edge [37,38].

6.4. Aggregation

Some simple API endpoints might only consist of requests to other external services like cloud databases or weather APIs. These requests can be moved from origin to edge servers and format the responses on edge [2]. Additionally, the responses of those endpoints might be cached on the edge server. More advanced handling can also be achieved using edge functions. Even transactional tasks can benefit from solely exchanging raw and compressed data between the origin and edge servers, where the final product is then assembled on edge [21].

6.5. Session Handling

Websites commonly keep sessions with clients to recognize users and keep temporary data across multiple requests. Often, the returned websites embed this temporary information based on the session token. For example, current shopping cart items can be associated with the session. Edge servers can be used to replace placeholders in a generic HTML site with the content acquired by the state. The other way around is also possible: If no session token is found, the edge can directly return the default site for anonymous users [20].

7. Conclusion and Future Work

CDNs are an essential part of accelerating web services and APIs through the caching of both static and dynamic content. They rely on a request-routing system to connect a user to an edge server and thus lower the main factor that contributes to a site's loading time: latency. We looked at the concept of virtual networks, which lower latency between the edge and origin servers and ways to accelerate dynamic content and APIs. The different methods of invalidation and cache control are tricky to facilitate correctly and demand careful consideration from developers with respect to the application build.

This paper also laid out how edge computing can be used to move different aspects of applications onto edge, like authentication, site assembly or endpoints based on accumulative requests. To conclude, these approaches can not only take load off the origin, lowering the bandwidth needed, but also make end user's experience better through faster loading times, especially when their request can completely be handled on edge.

To lessen the design complexity of edge functions for custom purging, research should be conducted to look at how invalidation can be done more automatically by using assumptions about data dependencies in the cached objects and API endpoints, and how unified API Guidelines can help in this endeavour.

References

[1] "Cloudflare Glossary - Internet exchange points," https://www.cloudflare.com/de-de/learning/cdn/glossary/internet-exchange-point-ixp/, [Online; accessed 28-March-2023].

[2] J. S. Erik Nygren, Ramesh K. Sitaraman, "The Akamai Network: A Platform for High-Performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1842733.1842736

[3] B. M. Maggs and R. K. Sitaraman, "Algorithmic Nuggets in Content Delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015. [Online]. Available: https://doi.org/10.1145/2805789.2805800

[4] "Why Akamai," https://www.akamai.com/why-akamai, [Online; accessed 28-March-2023].

[5] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. "O'Reilly Media, Inc.", 2013.

[6] G. Peng, "CDN: Content Distribution Network," 2004. [Online]. Available: https://arxiv.org/abs/cs/0411069

[7] H. Beheshti, "Fastly - Leveraging your CDN to cache "uncacheable" content," https://www.fastly.com/blog/leveraging-your-cdn-cache-uncacheable-content, [Online; accessed 28-March-2023].

[8] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari, "Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 1–12. [Online]. Available: https://doi.org/10.1145/511446.511448

[9] W. Wingerath, F. Gessert, E. Witt, H. Kuhlmann, F. Bücklers, B. Wollmer, and N. Ritter, "Speed Kit: A Polyglot and GDPR-Compliant Approach For Caching Personalized Content," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1603–1608. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00142

[10] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 531–537. [Online]. Available: https://doi.org/10.1145/2815675.2815717

[11] Z. Wang, J. Huang, and S. Rose, "Evolution and challenges of DNS-based CDNs," *Digital Communications and Networks*, vol. 4, no. 4, pp. 235–243, 2018. [Online]. Available: https://doi.org/10.1016/j.dcan.2017.07.005

[12] F. Chen, R. K. Sitaraman, and M. Torres, "End-User Mapping: Next Generation Request Routing for Content Delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 167–181, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787500

[13] "EdgeNext CDN Introduction," https://home.console.edgenext.com/#/doc/content/cdn/Product%20Introduction/Product%20overview, [Online; accessed 28-March-2023].

[14] "How CloudFront delivers content," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/HowCloudFrontWorks.html, [Online; accessed 28-March-2023].

[15] "Microsoft Azure Front Door - Traffic acceleration," https://learn.microsoft.com/en-us/azure/frontdoor/front-door-traffic-acceleration, [Online; accessed 28-March-2023].

[16] "Google Cloud CDN - Choose a CDN Product," https://cloud.google.com/cdn/docs/choose-cdn-product, [Online; accessed 28-March-2023].

[17] "Amazon CloudFront Key Features," https://aws.amazon.com/cloudfront/features/?nc1=h_ls&whats-new-cloudfront.sort-by=item.additionalFields.postDateTime&whats-new-cloudfront.sort-order=desc, [Online; accessed 28-March-2023].

[18] "How Microsoft builds its fast and reliable global network," https://azure.microsoft.com/en-us/blog/how-microsoft-builds-its-fast-and-reliable-global-network/, [Online; accessed 28-March-2023].

[19] "Microsoft CDN - Dynamic Site Acceleration," https://learn.microsoft.com/en-us/azure/cdn/cdn-dynamic-site-acceleration, [Online; accessed 28-March-2023].

[20] J. Anton, L. Jacobs, X. Liu, J. Parker, Z. Zeng, and T. Zhong, "Web Caching for Database Applications with Oracle Web Cache," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 594–599. [Online]. Available: https://doi.org/10.1145/564691.564762

[21] A. Davis, J. Parikh, and W. E. Weihl, "Edgecomputing: Extending Enterprise Applications to the Edge of the Internet," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters*, ser. WWW Alt. '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 180–187. [Online]. Available: https://doi.org/10.1145/1013367.1013397

[22] "Amazon CloudFront - Managing how long content stays in the cache," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html, [Online; accessed 31-March-2023].

[23] "Cloudflare CDN - Purge Cache," https://developers.cloudflare.com/cache/how-to/purge-cache/, [Online; accessed 31-March-2023].

[24] "Fastly - API Caching, Part 1," https://www.fastly.com/blog/api-caching-part-i, [Online; accessed 30-March-2023].

[25] M. Hossein Sheikh Attar and M. Tamer Özsu, "Alternative Architectures and Protocols for Providing Strong Consistency in Dynamic Web Applications," *World Wide Web*, vol. 9, no. 3, pp. 215–251, Oct. 2006. [Online]. Available: https://doi.org/10.1007/s11280-006-8563-1

[26] "Fastly - Purging," https://developer.fastly.com/reference/api/purging/, [Online; accessed 31-March-2023].

[27] "Fastly - API Caching, Part 2," https://www.fastly.com/blog/api-caching-part-ii, [Online; accessed 30-March-2023].

[28] "Fastly - API Caching, Part 3," https://www.fastly.com/blog/api-caching-part-iii, [Online; accessed 30-March-2023].

[29] "Amazon CloudFront - Controlling the Cache Key," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/controlling-the-cache-key.html, [Online; accessed 31-March-2023].

[30] "Wundergraph Architecture - Manage API Dependencies explicitly," https://docs.wundergraph.com/docs/architecture/manage-api-dependencies-explicitly, [Online; accessed 31-March-2023].

[31] "Hygraph - Caching," https://hygraph.com/docs/api-reference/basics/caching, [Online; accessed 31-March-2023].

[32] "AWS Lambda Features," https://aws.amazon.com/lambda/features/, [Online; accessed 31-March-2023].

[33] "Cloudflare - How Workers work," https://developers.cloudflare.com/workers/learning/how-workers-works/, [Online; accessed 01-April-2023].

[34] T. Berners-Lee, R. T. Fielding, and L. M. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, Jan. 2005. [Online]. Available: https://doi.org/10.17487/RFC3986

[35] Suresha and J. R. Haritsa, "On Reducing Dynamic Web Page Construction Times," in *Advanced Web Technologies and Applications*, J. X. Yu, X. Lin, H. Lu, and Y. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 722–731. [Online]. Available: https://doi.org/10.1007/978-3-540-24655-8_78

[36] "Google Cloud CDN - Using JWT to authenticate users," https://cloud.google.com/api-gateway/docs/authenticating-users-jwt, [Online; accessed 31-March-2023].

[37] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, "Management of API Gateway Based on Micro-service Architecture," *Journal of Physics: Conference Series*, vol. 1087, no. 3, p. 032032, sep 2018. [Online]. Available: https://doi.org/10.1088/1742-6596/1087/3/032032

[38] "Wundergraph - OpenID Connect-Based Authentication," https://docs.wundergraph.com/docs/features/openid-connect-based-authentication, [Online; accessed 31-March-2023].

Wireless Time Synchronization in IEEE 802.11

Ulkar Aslanova, Leander Seidlitz*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: ulkar.aslanova@tum.de, {leander.seidlitz, jonas.andre}@net.in.tum.de

***Abstract*—IEEE 802.11 Wireless Local Area Networks (WLANs) are extensively utilized for communication among multiple devices within a limited geographical area. In comparison to Ethernet-based networks, WLANs provide users with mobility and flexibility. However, wireless communication technologies have introduced new challenges, one of which is time synchronization, which is essential for network management and monitoring. While the synchronization issue has been extensively researched in conventional wired networks, the physical limitations of the wireless medium have presented a new set of difficulties. The lasting problem with the lack of synchronized timing support has been addressed in IEEE 802.11-2012. This amendment introduced two methods: Timing Advertisement (TA) and Timing Measurement (TM). This paper analyzes the time synchronization mechanisms included in the IEEE 802.11 standard, along with exploring non-IEEE solutions. It also examines the various factors that influence synchronization performance.**

***Index Terms*—IEEE 802.11 WLAN, Time Synchronization, Timing Synchronization Function (TSF), Timing Advertisement (TA), Timing Measurement (TM), Network Time Protocol (NTP), IEEE 1588 Precision Time Protocol (PTP)**

1. Introduction

Wireless communication has brought major changes to data networking and telecommunications. In today's world, wireless networks are widely employed in fields that formerly relied on traditional wired networks. One such network is the Wireless Local Area Network (WLAN), which offers users high bandwidth connectivity within a limited geographical area [1]. The adoption of IEEE 802.11 standards has had a significant influence on the both public and private domains. 802.11 has developed into a common option for a constantly growing application field due to its cost-effective chipsets and support for high data rates [2]. In modern times, WLANs are utilized to provide a communication infrastructure for a wide range of applications, spanning from small-scale in-home networks to large-scale deployments in office buildings, as well as mobile networks in airports and other public spaces.

1.1. Infrastructure Mode in IEEE 802.11

In IEEE 802.11, there are several communication modes defined for the transmission of data between devices. These communication modes control the interaction and information exchange among devices within the network. The basic service set (BSS) is a fundamental component of IEEE 802.11 WLAN. It is a group of wireless stations (STAs) that communicate at the physical layer (PHY). Depending on the communication mode, BSS can be classified into three categories: independent BSS (IBSS), infrastructure BSS and Mesh BSS (MBSS). This paper will focus on investigating time synchronization techniques designed for the infrastructure mode in IEEE 802.11-based networks.

The infrastructure mode of WLAN enables communication between STAs through a centralized entity called an access point (AP). The AP manages the network and coordinates communication between devices. This mode is particularly suitable for use in centralized architectures utilized in various applications, such as smart grids [3] and industrial automation [4].

1.2. Time Synchronization Problem

In an ideal clock, the rate remains constant over time, while an ordinary clock develops an offset. So, if an ordinary clock C_{ord} has a rate of a and an offset b to the ideal clock t, then C_{ord} can be calculated using (1).

$$C_{ord}(t) = a \cdot t + b \tag{1}$$

The goal of time synchronization is to minimize the error ϵ (2) between two clocks.

$$\epsilon = C_{ord}(t) - t \tag{2}$$

Time synchronization among wireless nodes is one of the key functions for controlling and monitoring activities within wireless networks. It is essential not only for network management but also for MAC layer protocols such as Time Division Multiple Access (TDMA), which relies on synchronized timing for achieving collision-free channel access in shared medium networks. Moreover, time synchronization is essential for power management in IEEE 802.11 networks [5], real-time applications [6], and Internet of Things (IoT) applications [7]. In IEEE 802.11, proper time synchronization enables power-saving features, allowing devices to coordinate wake-sleep schedules efficiently. For real-time applications, accurate time synchronization is crucial to minimize jitter and latency. In IoT applications, time synchronization is fundamental for synchronizing data from multiple sensors and devices.

Despite the fact that synchronizing wireless nodes is essential, there has not been much support for providing synchronized clocks until the introduction of the IEEE 802.11-2012 amendment. Prior to this amendment, the

 doi: 10.2313/NET-2023-11-1_12

preferred approach to achieve synchronized clocks in IEEE 802.11 networks was the utilization of synchronization protocols such as the Network Time Protocol (NTP) and the IEEE 1588 Precision Time Protocol (PTP) over WLAN. The 802.11-2012 standard has expanded the methods available for clock synchronization in wireless LANs with two mechanisms: Timing Advertisement (TA) and Timing Measurement (TM).

The remaining sections of the paper are structured as follows: Section 2 examines alternative clock synchronization protocols that are not part of the IEEE 802.11 standard; Section 3 provides an overview of the clock synchronization mechanisms included within the IEEE 802.11-2012 standard; and Section 4 analyzes the performance aspects of wireless time synchronization.

2. Non-IEEE 802.11 Protocols for Time Synchronization

Due to the lack of time synchronization support in IEEE 802.11, other protocols such as IEEE 1588 Precision Time Protocol (PTP) and Network Time Protocol (NTP) have been used for synchronization. These protocols were originally designed for synchronization purposes in wired networks and later adapted for wireless networks.

Both of these protocols can be utilized for relative and absolute time synchronization. The goal of absolute synchronization, also known as external synchronization, is to align devices within a network to a universal reference such as International Atomic Time (TAI) or Universal Coordinated Time (UTC). This ensures that all devices maintain a consistent time with respect to the specified reference. Relative synchronization (or internal synchronization) is employed solely to establish a shared timebase among synchronized devices within a network.

2.1. PTP over WLAN

PTP, introduced in the IEEE 1588 standard, is commonly used in LAN networks. It is based on a master-slave approach, where the master node is responsible for synchronizing slaves in the network. In PTP, the default Best Master Clock Algorithm (BMCA) is used. In infrastructure mode WLAN, the AP can be considered as the master and other STAs as slaves. Therefore, a custom BMCA can be implemented, which will choose AP as the master clock [8]. Several ways have been researched in order to employ PTP over WLAN. The prototypes using software and hardware-based timestamping were first presented in [9].

PTP uses a two-way packet exchange for synchronization. The master clock periodically sends synchronization messages (SYNC) every two seconds (by default), which contain an estimated timestamp of the message transmission time t_1. Upon receiving the SYNC message, the slave stores a timestamp of the reception time t_2. The master clock can also send a follow-up message with a more precise value of the transmission time timestamp. The difference $t_2 - t_1$ could be used for calculating the offset; however, it includes not only the offset from slave to master o_{sm}, but also the propagation delay d_{ms} from master to slave (3).

$$t_2 = t_1 + d_{ms} + o_{sm} \tag{3}$$

For this reason, the slave clock periodically sends a delay request (DELAY_REQ) message to the master clock and records the transmission time timestamp t_3. In response to the request, the master clock sends a delay response (DELAY_RESP) message containing the reception time of the received request message t_4 [9]. The difference t_4 - t_3 includes the propagation delay d_{sm} from slave to master (4).

$$t_4 = t_3 + d_{sm} - o_{sm} \tag{4}$$

With this information, the slave can calculate the offset to the master o_{sm} using (5), and adjust its clock accordingly [8].

$$o_{sm} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} - \frac{d_{ms} - d_{sm}}{2} \tag{5}$$

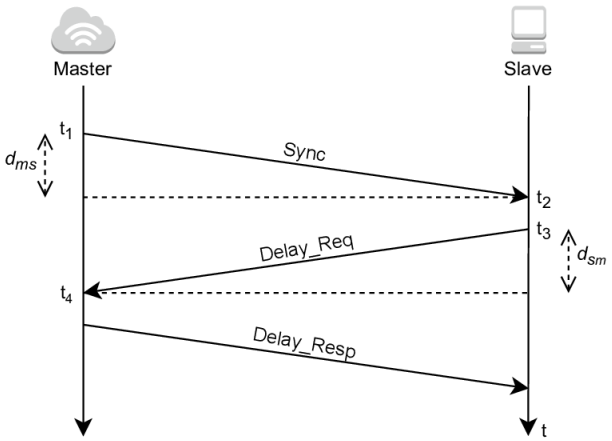


Figure 1: Message exchange in PTP [10];

If the propagation delays d_{ms} and d_{sm} are equal, then it is possible to calculate the offset precisely. PTP assumes that the propagation delay is symmetric in both directions, but this assumption is not always true and can result in synchronization bias. As shown by Mahmood et al. in [11], in wireless networks, asymmetry comes from the multicasting of packets from STA to AP, and the propagation delay in the direction from slave to master is commonly greater than from master to slave.

One of the main problems that arises when using PTP over WLAN is the handover of STAs from one AP to another, which is one of the requirements for use in large industrial environments where mobility is important. As PTP is designed for wired networks, it does not provide a fast handover of slaves from one master to another. A broader discussion of this problem has been presented in [11].

2.2. NTP over WLAN

NTP is a client-server protocol which can be used for time synchronization over WLAN. In client-server synchronization clients should request synchronization from the server. In a WLAN setup, the AP can be seen as a server, while the STAs function as clients. NTP uses a two-way packet exchange and can calculate the offset using (5). Similar to PTP, NTP also makes the assumption of symmetric propagation delays.

 doi: 10.2313/NET-2023-11-1_12

Timestamping in NTP can be implemented using both software-based and hardware-based approaches. In the case of software timestamping, the performance depends on where the timestamping is done. When timestamping is done at the application level, the timestamping jitter can increase due to random channel access delays. Timestamping can also be done in the device driver to avoid channel access delays. Using hardware timestamping can provide more accurate and reliable timestamps [8].

3. Synchronization over IEEE 802.11

Various synchronization methods included in IEEE 802.11 are shown in Figure 2.

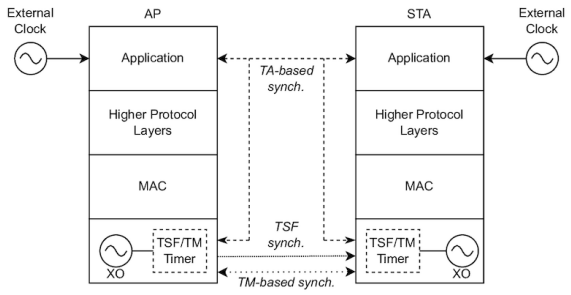


Figure 2: Synchronization Schemes in IEEE 802.11 [7];

3.1. Relative Synchronization Methods in IEEE 802.11

As all STAs communicate through the AP in infrastructure mode, it is essential that their clocks are synchronized with the AP. For internal synchronization, IEEE 802.11 uses the Timing Synchronization Function (TSF) timer with modulus 2^{64}, counting in increments of microseconds, which is present within every STA. In order to synchronize other STAs in a BSS, AP sends special frames called Beacon frames. These frames contain the TSF timer of the AP. STAs receive Beacon Frames at a regular rate, with the interval determined by their dot11BeaconPeriod parameter. The value of this parameter is included in Beacon frames and is defined by the AP. When a STA joins the BSS, it should adjust its beacon period to match the one specified in the Beacon frame [12]. As soon as a STA receives the Beacon frame, it should accept the timing information contained within this frame. If the STA's TSF timer differs from the received timestamp, it should update its local timer by adding the delay introduced by the STA's local PHY components and the time elapsed since the first bit of the timestamp was received at the MAC/PHY interface to the received timestamp.

It should be noted that the TSF method applies only offset correction and does not include rate correction. Additionally, it does not estimate propagation delay when calculating the offset. Both of these can lead to synchronization bias. However, the accuracy requirement for the TSF timer in IEEE 802.11 is achievable even without performing propagation delay compensation [8]. As the TSF timer's accuracy shall be within a tolerance of ±0.01%, it can still be met using hardware timestamps from TSF timers.

3.2. Absolute Synchronization Methods in IEEE 802.11

Alongside relative synchronization, IEEE 802.11-2012 has introduced two external synchronization mechanisms.

3.2.1. Timing Measurement (TM) Method. The TM method employs a two-way packet exchange between STA and AP for end-to-end synchronization. To initiate the Timing Measurement Procedure, the STA sends a request to the AP in the form of an Action frame, with the trigger value set to 1. Then, the AP starts transmitting action frames. The initial frame is transmitted by the AP at time t_1 and received by the STA at time t_2. Upon receiving this frame, the STA promptly sends an acknowledgement frame at time t_3. The AP receives this frame at time t_4 and responds with an action frame containing the values t_1 and t_4.

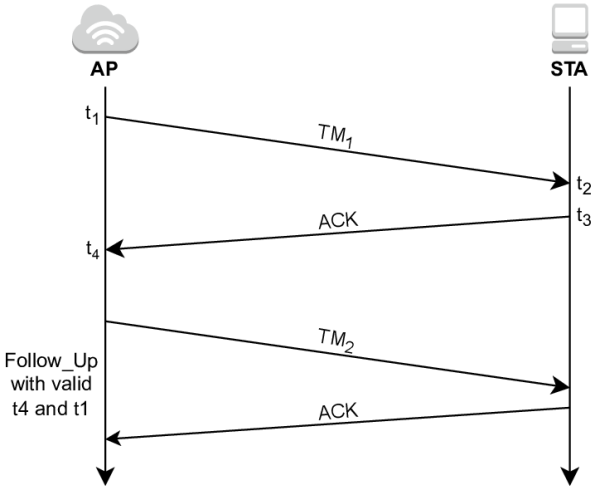


Figure 3: Message exchange in TM method [13];

With these values, the STA can calculate its time offset O from the AP using (5). Now, using the computed offset and the time reference provided by the AP, the STA can synchronize its own clock with the reference clock. To stop the timing measurement procedure, the STA should send a new action frame with the trigger value set to 0 [12].

In this method, the timestamping timer has a resolution of 10 ns, which is different from the TSF timer used in IEEE 802.11 networks, with a resolution of 1 µs. The IEEE 802.11 standard does not provide specific details regarding the timer used for timestamping in the TM method, so it's assumed that this timer is implemented as a vendor-specific timer and is capable of carrying TAI or UTC time [14].

3.2.2. Timing Advertisement (TA) Method. The TA method uses the TSF timer of the AP along with external timing standards such as UTC or TAI to achieve synchronized time across the STAs within a BSS. The AP shares a timestamp from its TSF timer and the offset between the TSF and the local clock (system clock). When

a STA receives a frame containing TA information, it generates a timestamp using its TSF timer. Then, it passes this timestamp and the received TA information to higher layers in order to synchronize its local clock with the AP's local clock.

Since the local timer and the TSF timer of the AP operate on different oscillators, the offset between them can change due to the skew between the two oscillators, which can affect synchronization. Also similarly to internal synchronization with TSF, the TA method can also lead to synchronization bias as it does not include the propagation delay between the sender and receiver. In [15] Mahmood et al. performed a performance analysis of the TA method.

4. Factors Influencing Synchronization Performance

Time synchronization performance is affected by several factors, such as the oscillator, quality of timestamps, clock adjustments, and synchronization rate. In this section, the effects of the mentioned factors will be discussed.

4.1. Oscillator Impact

A clock consists of an oscillator, which serves as the fundamental source of clock ticks. The main time source in modern-day devices is a quartz crystal oscillator (XO). In an ideal clock, the frequency of the oscillator remains constant over time. However, XOs frequency stability and accuracy are affected by various physical and electrical factors such as temperature, voltage, noise, and other conditions [16]. In order to achieve high accuracy and precision, it is important to minimize the error originating from the oscillator.

4.2. Timestamping Accuracy

In packet-based networks like IEEE 802.11, timestamps can be included in packets to distribute time information throughout the network. An important factor is when and how the timestamps are drawn. To ensure accurate offset calculation and avoid asymmetry, the timestamps should be drawn based on a common reference point, for example, upon detecting the start of the frame delimiter or packet preamble [8]. There are two methods for drawing timestamps: hardware-based and software-based. Hardware-based timestamps are generated by the hardware at the physical (PHY) layer and are known for their high accuracy. The TSF and TM methods utilize hardware timestamps. In contrast, software timestamps, typically generated within the device's operating system (OS) at higher protocol layers, are generally less accurate as they do not provide the exact departure and arrival times of packets. When using software timestamping, it is crucial to employ a stable clock in order to minimize the variation in access time when deriving timestamps. One important aspect is determining the location for software timestamping. As Mahmood et al. mentioned in [8], the device driver is often the initial point where software timestamping can be implemented. The timestamp can be captured within the interrupt service routine (ISR) in the driver, as this is the earliest point in time when the operating system (OS) is notified of an incoming or outgoing packet. The TA method is one of the software-based approaches for time synchronization. Table I in [7] presents the timestamping type and achievable accuracy for each of the mentioned methods.

4.3. Clock Adjustment

The clock adjustment is performed when a STA synchronizes its local clock with the reference clock, which is the AP in infrastructure mode. This adjustment aims to minimize the offset between the STA's clock and the reference clock. Various approaches can be employed for this process, including linear least-squares regression, statistical methods, and control theory-based approaches. Depending on the chosen method, an adaptive methodology should be implemented to account for the variations in error sources [8].

4.4. Synchronization Rate

The synchronization rate determines the frequency at which synchronization packets are exchanged between the AP and the STAs. The goal of selecting an appropriate synchronization rate is to minimize the combined effect of errors introduced by the oscillator, propagation delays, and other factors. Hardware timestamps, when used with a higher synchronization rate, can lead to higher accuracy in time synchronization as they rely on hardware components that provide precise and consistent timing measurements. On the contrary, software timestamps may not effectively improve accuracy with higher synchronization rates, as the software will frequently collect timestamps, including those with potential noise and jitter. Bringing noisy timestamps into the controller more frequently can result in inaccurate time measurements and introduce instability in the synchronization process [13]. However, it's important to note that lower synchronization rates can result in larger deviations from the reference clock, so finding the right balance is essential.

5. Conclusion and Future Outlook

This paper analyzed different time synchronization methods for the IEEE 802.11 infrastructure mode. This work presented non-IEEE protocols such as PTP and NTP and examined their applications in wireless networks. However, it is important to note that these are not the only non-IEEE solutions available. There are some other custom designed protocols. For example, in [15], a new protocol called SyncTSF was introduced, which provides relative time synchronization. Additionally, this paper discussed the new methods introduced in the IEEE 802.11-2012 amendment. The utilization of the TSF scheme in ad-hoc and mesh BSS modes can be found in [7]. Furthermore, different factors that have an impact on time synchronization have been presented in this work. It is essential to consider all of the factors introduced in order to achieve high performance.

Future work may include the development of protocols that can handle propagation delays. Also, it should be noted that fault tolerance and robustness should be considered in future development.

References

[1] B. Crow, I. Widjaja, J. Kim, and P. Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[2] G. R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X. P. Costa, and B. Walke, "The IEEE 802.11 universe," *IEEE Communications Magazine*, vol. 48, no. 1, pp. 62–70, 2010.

[3] P. P. Parikh, M. G. Kanabar, and T. S. Sidhu, "Opportunities and challenges of wireless communication technologies for smart grid applications," in *IEEE PES General Meeting*. IEEE, 2010, pp. 1–7.

[4] S. Ivanov, E. Nett, and S. Schemmer, "Automatic WLAN localization for industrial automation," in *2008 IEEE International Workshop on Factory Communication Systems*. IEEE, 2008, pp. 93–96.

[5] J. Singh, "PERFORMING CLOCK SYNCHRONIZATION FOR POWER MANAGEMENT IN MULTI-HOP AD HOC NETWORKS," *International Journal on Intelligent Electronic Systems*, vol. 3, 01 2009.

[6] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, 2000, pp. 125–132.

[7] A. Mahmood, T. Sauter, H. Trsek, and R. Exel, "Methods and performance aspects for wireless clock synchronization in IEEE 802.11 for the IoT," in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. IEEE, 2016, pp. 1–4.

[8] A. Mahmood, R. Exel, H. Trsek, and T. Sauter, "Clock Synchronization Over IEEE 802.11—A Survey of Methodologies and Protocols," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 907–922, 2017.

[9] J. Kannisto, T. Vanhatupa, M. Hannikainen, and T. Hamalainen, "Software and hardware prototypes of the IEEE 1588 precision time protocol on wireless LAN," in *2005 14th IEEE Workshop on Local & Metropolitan Area Networks*. IEEE, 2005, pp. 6 pp.–6.

[10] A. Mahmood, R. Exel, and T. Sauter, "Delay and Jitter Characterization for Software-Based Clock Synchronization Over WLAN Using PTP," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1198–1206, 2014.

[11] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kerö, "Towards high accuracy in IEEE 802.11 based clock synchronization using PTP," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2011, pp. 13–18.

[12] "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999*, pp. 1–1076, 2007.

[13] A. Mahmood, R. Exel, and T. Sauter, "Impact of hard-and software timestamping on clock synchronization performance over IEEE 802.11," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE, 2014, pp. 1–8.

[14] J. Henry, "Indoor Location : study on the IEEE 802.11 Fine Timing Measurement standard," Ph.D. dissertation, Ecole nationale supérieure Mines-Télécom Atlantique, 2021.

[15] A. Mahmood, R. Exel, and T. Sauter, "Performance of IEEE 802.11's Timing Advertisement Against SyncTSF for Wireless Clock Synchronization," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 370–379, 2017.

[16] H. Zhou, C. Nicholls, T. Kunz, and H. Schwartz, "Frequency Accuracy & Stability Dependencies of Crystal Oscillators," 2023.

Machine Learning Applications In 5G Network Orchestration

David Friedlein, Philippe Buschmann*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: david.friedlein@tum.de, phil.buschmann@tum.de

***Abstract*—The ability to virtualize and separate multiple networks on top of a common physical infrastructure allows network providers to serve different needs. With this approach, 5G networks can better support new technologies such as self-driving cars, which is not possible with traditional one-size-fits-all architecture. This is possible while also reducing the cost for the operators. The drawback is that it requires a lot of configuration and management to function optimally. Machine learning is a possible solution to simplify and automate this work.**

This paper analyses and compares multiple different proposed implementations of machine learning in the network slicing process. We see that all approaches provide benefits but they can not be directly compared to each other, because the measurements are too different.

***Index Terms*—5G, network slicing, machine learning, software-defined networks**

1. Introduction

Earlier mobile standards like 4G and 3G are mainly designed for smartphones and thus have a design solely focused on this purpose. With ongoing technological development, new and different use cases arise. These include self-driving cars, telemedicine and Internet of Things (IoT). All use cases for mobile networking can be divided into three classes following specifications from the International Telecommunication Union (ITU):

- **enhanced mobile broadband (eMBB)** Mainly meant for smartphones which need high data rates and a large area covered due to their mobility.
- **massive machine type communication (mMTC)** Sporadical communication of a large number of devices in a small area. It is used for IoT devices like sensors, which only send small amounts of data in large timeframes. Packet loss is not a significant problem.
- **ultra reliable low latency communication (uRLLC)** Communication with access over 99.9999% and end-to-end latency of less than 50 ms is required for some industrial use cases. For example smart connected fabrication plants.

For every class the network has to fulfill different needs. Serving all of these classes with one network while providing a consistent quality of service (QoS) is hard to achieve. A potential solution could be to build multiple different radio access networks, each specialized for one class. However, the development of multiple networks leads to a high amount of additional costs for network providers.

A better solution is network slicing. It allows network providers to use one physical network to serve all traffic classes while still providing a consistent QoS. It works by separating the network into multiple network slices (NS), which are tailored to a specialized purpose.

The management of all these slices can be quite complicated. Machine learning can simplify and automate this management.

The remainder of this paper analyzes three different approaches. Section 2 explains the background of network slicing and the enabling technologies. Section 3 explains three different papers and their results, which are then discussed in Section 4. Section 6 concludes the paper.

2. Background

In this section we explain the key technologies that enable network slicing.

2.1. Network Function Virtualization

Network function virtualization (NFV) is a concept that enables virtualization to separate hardware from functionality [1]. Network functions (NF) like virtual firewalls or virtual load balancers can be deployed on servers to run these network functions on any server. This helps create flexible networks by deploying necessary NFs on servers when needed. It reduces the dependency on special hardware and the placement of servers in the network.

This technology is crucial for network slicing

2.2. Software Defined Networking (SDN)

Software defined networking (SDN) physically separates the network control plane from the forwarding plane [2]. The forwarding plane consists of all the hardware that forwards packets while the control plane consists of one or more SDN controllers. The control plane has knowledge about the whole network and its policies. With this information, it makes all the routing decisions and communicates them via different protocols to the forwarding plane. Then routers and switches follow the decisions of the control plane and forward the packets. In comparison to a classical network where every router makes its own decisions about forwarding packets, SDN centralizes the routing process. These differences allow

 doi: 10.2313/NET-2023-11-1_13

the network a faster adaption to changes in the network or the users' needs.

SDN is necessary for network slicing to adjust the routing decisions to changing network slices and their different routing needs.

2.3. Network Slicing

Using the above explained technologies modularizing networks is possible. The NGMN (Next Generation Mobile Network) has introduced network slicing for 5G in [3]. It allows the creation of multiple logically independent networks that operate on top of a unified physical infrastructure. Network providers can customize these logical networks to provide different services and performance levels to meet the demand of multiple clients at the same time.

For the creation of these slices three required layers were defined by the NGMN [3]:

- The **infrastructure resource layer** comprises all the physical resources, which includes access nodes, cloud nodes, end-user devices such as smartphones and wearables and even the links between these devices. All devices have different capabilities and can fulfill different roles in the network. These capabilities and roles can be controlled and monitored through an application programming interface (API).
- The **business enablement layer** contains all of the functions and configuration parameters of the network devices. Some of these functions offer different levels of performance, which are used to differentiate the network slices. They are separated into modular blocks and can be loaded onto required devices by an API.
- The **business application layer** contains all applications and services provided by the network operator or different enterprises.

The 3 layers are connected by the **E2E management and orchestration entity**. This entity controls the creation, scaling and geographic distribution of resources of all network slices. It defines a slice depending on the use case and applications needed. It chooses the required network functions with specific performance levels to map those onto the device in the infrastructure resource layer. It makes decisions about the scaling for the lifetime of the slice. It shifts resources between slices to optimize performance.

2.4. 3GPP Specification

The 3GPP defines a management and orchestration architecture. The communication service management function (CSMF) translates incoming requests for services into requirements for the network. These requirements are sent to the Network Slice Management Function (NSMF), which chooses or generates the slice blueprint optimal for the requirements. A slice blueprint contains all needed NFs, their connections and configurations. After the slice is instantiated the NSMF manages it until it is decommissioned.

A Network Slice instance (NSI) is a group of NFs. "An NSI is composed of NFs shared between two or more slices, as well as dedicated NFs" [4].

3. Network Slicing with Machine Learning

In recent years a lot of research about the usage of ML for resource orchestration has been conducted. Multiple research groups have proposed different methodologies to include machine learning in the decision process. Some of these are explained in the following.

3.1. Artificial Intelligence for Slice Deployment and Orchestration

Dandachi et al. [4] propose two new approaches for ML based on the 3GPP NSMF architecture. They define a novel architecture that is compatible with the 3GPP design and includes three new functions:

The slice analytics (SA) function minimizes required resources by sharing them between slices. If multiple slices want to use the same NF they can be grouped in a common NSI.

The admission control (AC) decides whether new slices can be created or have to be dropped because of resource shortage.

This function can be combined with the congestion control (CC) function, which scales slices up and down as needed, to build a cross-slice admission and congestion control (CSACC).

3.1.1. Slice Analytics (SA). The two main tasks of the SA are the classification of slices and the reduction of required resources. For this purpose, it receives the slice blueprint and the resource requirements for new slices. If the requested slice requires NFs and resources that are already used by other slices, they can be shared between the two slices. This reduces the number of new resources that have to be allocated, which then reduces the rate of denied requests due to a shortage of available resources.

Depending on the specific slices and services running on them, the amount of NFs that can be shared is different. This allows the classification into elastic and non-elastic slices [5]. The authors of [4] only consider elastic slices in their research. The usage is explained in the setting of a sports event: Different broadcasts will use the same images, which can be shared, but will provide different commentary, which has to be separated.

The performance of two different algorithms for the grouping are analysed. The first algorithm finds an existing NSI with the highest amount of overlapping NFs using the Jaccard similarity. For each new slice, the best group can be calculated and only the missing NFs are created and added to the NSI.

The second algorithm uses spectral clustering [6] to create NSIs that reduce resource usage. This algorithm does not find an existing NSI to which new slices fit but calculates an optimal grouping for all existing and new slices. This has a the higher complexity ($\mathcal{O}(N^3)$) [6] compared to calculating the Jaccard similarity ($\mathcal{O}(N)$) and thus the authors recommend executing this recalculation in larger time intervals or when the system is overloaded.

Dandachi et al. classify all slices into either guaranteed quality-of service (GS) slices, which have a high priority, or best effort (BE) slices, which have a lower priority. Additionally, if the system needs more resources for GS slices these can be taken from BE slices. For each class, a queue exists, in which new slice requests are inserted until they are created. After the new slice request has been grouped with other slices it is inserted into one of two queues depending on the class of the slice.

3.1.2. Cross-slice admission and congestion control (CSACC). The CSACC function decides which queued slices are accepted and how many resources get assigned to the slices. The goal is to maximize the number of accepted slices while reducing the probability that a new slice request has to be dropped because the queue is full. The resources of each slice can not be reduced beyond a minimum level to prevent extreme degradation in the QoS.

For this, reinforcement learning is employed. State-Action-Reward-State-Action (SARSA) aims to find a policy that maps the state of the system to an action, which maximizes the reward. To enhance this model the authors use linear function approximation.

3.1.3. Performance results. Comparing only the slice admission with SARSA to the CSACC with SARSA, the latter method shows an improvement with up to 23% reduction in dropped slice requests. However, this improvement is only possible if new GS slices are requested with a probability of less than 70%. Higher values show no difference between the two methods.

Using the SA function, the rate of dropped slices is even further reduced. Up to 44% reduced drop rate is achieved by using CSACC with SARSA and SA with spectral clustering compared to not using any function, as seen in Figure 1.

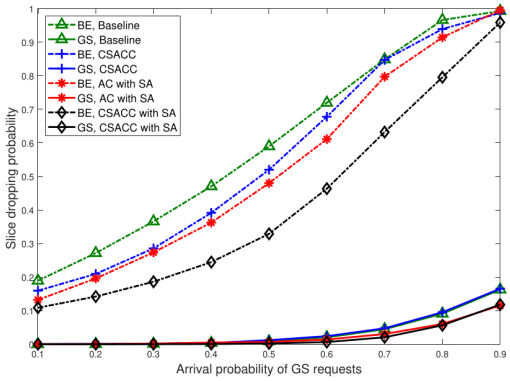


Figure 1: Slice dropping rate as a function of arrival probability from [4]

3.2. Artificial Intelligence for Elastic Management

In a paper from Gutierrez-Estevez et al. [7] the concept of resource elasticity (which was defined in an earlier paper from the same group) is being used as the basis of multiple ML approaches. Resource elasticity describes the ability of a network to automatically and smoothly adapt to changes in the system. This elasticity can be applied to three areas:

- **computational elasticity** in the operation of VNFs
- **orchestration-driven elasticity** in the placement of VNFs
- **slice-aware elasticity** in the distribution of resources between slices

3.2.1. Computationally Elastic Scheduler. One of the more computationally expensive NF is the media access control (MAC) scheduler. It is responsible for assigning bandwidth resources to different devices in a network and deciding on which modulation and coding scheme (MCS) to use. Depending on the signal-to-noise ratio (SNR) in the connection, different MCSs are best suited and have different computational complexities. Contextual bandits are an ML approach that tests different randomized policies. The policies are finetuned regarding environmental conditions. The predicted SNR for a given user is a necessary condition for the contextual bandits. Applying a long short term memory network to this predicts the SNR.

3.2.2. Slice-Aware Resource Management. The authors of [7] design algorithms, which are supposed to optimally allocate/de-allocate resources to individual slices. They have to consider QoS requirements, Service Level Agreements (SLAs) and demands of the slices. These algorithms can be applied to different problems.

The authors use a deep neural network to forecast the amount of traffic in the future. This helps to allocate additional resources, if a large group of users, which increases the demand, is predicted, or de-allocate resources at day times when little to no traffic is expected. Algorithms to adjust different settings in the network can also be improved with this data.

Predicting the movement of people helps to adjust the settings of cell towers such as the beam pattern or even allocate towers at different locations to the slice. Identifying groups of people and predicting their movement can be done by ML algorithms. The position and demand of users are necessary to guarantee reliable coverage.

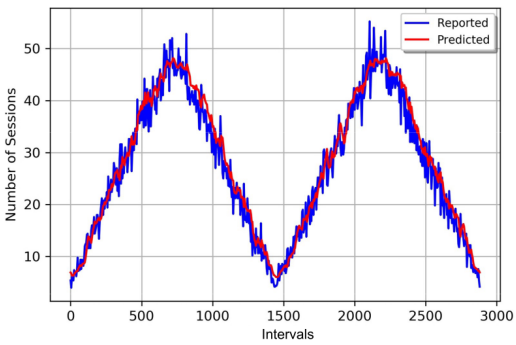


Figure 2: Traffic prediction results from [7]

3.3. Machine Learning Based Resource Orchestration for 5G Network Slices

The group of Salhab et al. [8] proposes a novel architecture that includes ML in the management and

 doi: 10.2313/NET-2023-11-1_13

orchestration process.

3.3.1. System design. The proposed network architecture is not based on the 3GPP architecture but contains four other components as seen in Figure 3. The first part is the **gatekeeper**. To aggregate and sort traffic into the correct slices, first a marking and classification phase is needed. Using the slice blueprints and tenant requests, the gatekeeper generates requirements using supervised learning. This allows us to choose the correct blueprint for each requested slice.

These policies are then handed to the **decision maker**, which is composed of a **forecast aware slicer** and an **admission controller**. The forecast aware slicer uses regression trees to predict the required ratio of all network slices. It achieves this using different information about the traffic. Using this and the current load on the network, the admission controller decides whether to grant requests for new slices or not.

If the request is accepted it gets sent to the **slice scheduler**. Its purpose is to find a schedule that serves all slices and minimizes the total time needed. Salhab et al. prove this to be an NP-hard problem [9] and provide a heuristic for solving it.

Denied requests are sent to the **resource manager**, which uses micro-services to automatically scale resources. If additional resources are needed to allow a new slice to be accepted it reduces the resources for other slices. Available resources can be assigned to slices, increasing their performance. The authors use Reinforcement learning to optimize decision making and improve the system's utilization.

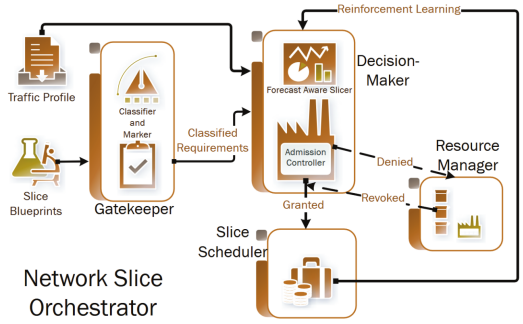


Figure 3: Block diagram of the architecture from [8]

3.3.2. Performance. The paper compares different machine learning algorithms running on the above-mentioned architecture. The first benchmark compares different models used for the classification. Most of the models achieve a prediction accuracy of over 90%. The highest accuracy of 98% was produced using a linear discriminant model.

The second benchmark compares different models used to predict the optimal slice ratios. Different tree algorithms (simple trees, medium trees and complex trees) achieved the lowest root mean squared error (around 5%) and the highest prediction speeds. These results, however, come at the expense of a longer training time compared to linear models. Since the models' training is infrequent, this tradeoff can be accepted. To validate the usefulness of the tree models, the authors compare them to the theoretical optimum, a static slice ratio and a random slice ratio. The ML model performed the best, with an average 5% gap to the theoretical optimum. The random approach performed the worst with a 30% gap.

For the last test, the setup was run with and without traffic forecasting. Using the forecasting the throughput of the system increased by approximately 30%.

4. Evaluation

All approaches are shown to be beneficial in some aspect. But comparing them against each other is difficult because different papers use ML to improve other aspects of the system. Moreover, all of the approaches analyzed in this paper measured different metrics of the network or the ML models. The authors of [7] only show the accuracy of the ML models, but not any performance results obtained from implementing them. Paper [4] focuses on the probability, that a new slice request has to be dropped. The last paper [8] includes the ML accuracy and the throughput of the system with the implemented methods.

Some aspects of an approach can not be measured with numbers. For example, the system architecture in [4] is based on the 3GPP architecture and the system in [8] introduces a completely new architecture. Both provide certain advantages and disadvantages. A standardized system makes it easier to expand and compare to other systems on the same architecture. Creating a new architecture allows the system to be better specialized for a certain use case.

Some current problems in this research area include the lack of data for training supervised models. Because 5G is not widely deployed and the hardware is expensive, collecting real-world data for training is difficult.

5. Related work

There have been multiple other surveys about this research area, which focus on certain types of approaches. Some focus on the applications of 5G and ML for IoT devices Wijethilaka et al. [10], Khan et al. [11]. Others concentrate on deep reinforcement learning Hurtado Sánchez et al. [12]. The survey from Su et al. [13] concentrates on mathematical models.

6. Conclusion and future work

In this paper, we examine different approaches to utilizing ML in the management and orchestration process for network slicing.

We found that ML seems to improve many aspects of the management and orchestration process. One architecture has been shown to increase the throughput of the system [8] other designs increase the number of slices that can be run on a system.

A possible solution to better compare different approaches would be to implement different designs on the same setup and measure the same parameters. This could be done in future work.

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," vol. 18, no. 1, 2016, pp. 236–262.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," vol. 17, no. 1, 2015, pp. 27–51.

[3] "NGMN 5G white paper," 2015.

[4] G. Dandachi, A. De Domenico, D. T. Hoang, and D. Niyato, "An Artificial Intelligence Framework for Slice Deployment and Orchestration in 5G Networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 858–871, 2020.

[5] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[6] S. Tsironis, T. Mauro Sozio, and M. Vazirgiannis, "Accurate Spectral Clustering for Community Detection in MapReduce," 2013.

[7] D. M. Gutierrez-Estevez, M. Gramaglia, A. D. Domenico, G. Dandachi, S. Khatibi, D. Tsolkas, I. Balan, A. Garcia-Saavedra, U. Elzur, and Y. Wang, "Artificial Intelligence for Elastic Management and Orchestration of 5G Networks," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 134–141, 2019.

[8] N. Salhab, R. Rahim, R. Langar, and R. Boutaba, "Machine Learning Based Resource Orchestration for 5G Network Slices," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[9] N. Salhab, R. Rahim, and R. Langar, "Throughput-Aware RRHs Clustering in Cloud Radio Access Networks," in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, 2018, pp. 1–5.

[10] S. Wijethilaka and M. Liyanage, "Survey on Network Slicing for Internet of Things Realization in 5G Networks," vol. 23, no. 2, 2021, pp. 957–994.

[11] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han, and C. S. Hong, "Network Slicing: Recent Advances, Taxonomy, Requirements, and Open Research Challenges," vol. 8, 2020, pp. 36 009–36 028.

[12] J. A. Hurtado Sánchez, K. Casilimas, and O. M. Caicedo Rendon, "Deep Reinforcement Learning for Resource Management on Network Slicing: A Survey," *Sensors*, vol. 22, no. 8, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/3031

[13] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu, "Resource Allocation for Network Slicing in 5G Telecommunication Networks: A Survey of Principles and Models," *IEEE Network*, vol. 33, no. 6, pp. 172–179, 2019.

Survey On The Current State Of Tor Over QUIC

Mohamed Mehdi Gharam, Lion Steger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: mehdi.gharam@tum.de , stegerl@net.in.tum.de

***Abstract*—Despite its popularity, Tor is currently bugged with high latency and other performance issues. In response, a transition in Tor's transport layer protocol to QUIC has been proposed, aiming to solve many of the problems caused by TCP's inherent limitations.**
In this paper, we present a literature survey on the current state of Tor over QUIC. We examine proposed designs and evaluate their impact on performance and security. We conclude that transitioning Tor to QUIC holds significant potential, yet further work on fully examining the security and performance impacts of such a transition still remains to be done.

***Index Terms*—Tor, QUIC**

1. Introduction

Anonymity networks have become a very important tool to face the increasing trend of tracking users and infringing on their privacy rights. Tor is the most popular anonymity network, currently used by over 3 Million clients [1]. Yet, it has long suffered from performance problems that stood in the face of its adoption. Many of the underlying causes, such as Head-of-Line blocking, unfairness in bandwidth allocation, and inefficient congestion control are caused by multiplexing Tor circuits over TCP connections [2]. While a lot of research has been done on improving Tor, most attempts have failed because they either compromise security or introduce additional performance overhead [3]. This motivates the choice behind a change in Tor's transport layer protocol to the UDP-based QUIC.
In this paper, we present two proposed designs for a Tor over QUIC implementation: an end-to-end design proposed by the Tor community that aims to improve end-to-end congestion control, and a hop-by-hop design adopted by the research community that aims to use QUIC's features to improve fairness and decrease latency. We then proceed to evaluate these designs from a security and performance perspective while giving an overview of the current state of research in the process.

2. Background

In this section, we introduce the most important concepts discussed in this paper. We give a brief overview of how Tor works, introduce the QUIC communication protocol, and go over the reasons behind changing Tor's transport layer to QUIC.

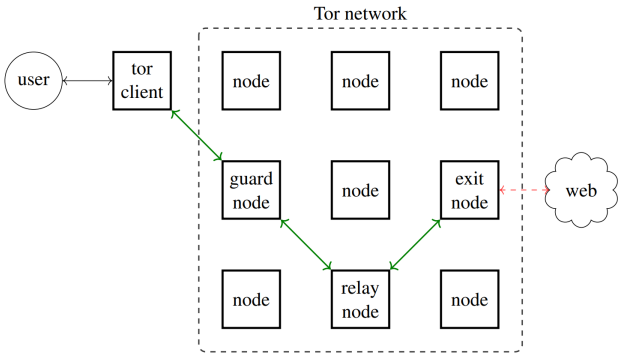


Figure 1: A visualization of a Tor circuit. Taken from [5].

2.1. TOR

Tor is a low-latency anonymity network that allows users to communicate online without exposing their identity [4]. It does this by relaying all traffic over multiple nodes in a process known as *Onion Routing*. Onion Routing allows the anonymisation of TCP traffic by rerouting it over multiple volunteer relays, known as *Onion Routers* (OR). By default, Tor uses three of these routers to build what is called a *circuit*. A circuit is first established when a client, also called an *Onion Proxy* (OP), attempts to connect to a server and proceeds to pick 3 Onion Routers: *a guard node, a middle (or relay) node,* and *an exit node*, as shown in Figure 1.
A TLS-protected TCP connection is then formed between each part of the circuit and the node directly following it. Each node of the circuit only knows its immediate predecessor and successor, making it so that only the guard node is aware of the client's identity and only the exit node knows the server. The middle node then ensures that the guard and exit nodes are unaware of each other. To further guarantee anonymity, messages are encrypted multiple times using pre-exchanged symmetric keys with the onion routers, with each OR adding or only removing its own layer of encryption so that messages are only revealed on the last node, by which time the sender's identity is already anonymous.
It's important to mention that there's only one TCP connection between Tor nodes. This means that circuits belonging to different clients are multiplexed over the same OR-OR connection. This has the obvious performance benefit of saving the time needed to establish such a connection every time it is needed, and it also increases the security of the network, as it makes it significantly harder to identify traffic of different circuits if the connection

 doi: 10.2313/NET-2023-11-1_14

is compromised. On the other hand, Tor also multiplexes several end-to-end TCP streams belonging to the same OP into the same circuit since establishing a circuit for each stream would increase latency.

2.2. QUIC

QUIC is a transport layer network protocol based on UDP, developed by Google as a new alternative to TCP [6]. It aims to solve many of the problems caused by TCP and comes with several features, such as integrated TLS1.3 encryption, user space implementation, and middlebox resistance through encryption of packet headers, making it easier to roll out updates. It also attempts to improve latency through decreasing handshake delay by reducing the required round-trip time (RTT) to establish a connection. Since security is an integral part of Tor, it is important to mention that the offered 0-RTT handshake raises some security concerns, as extra protection against replay attacks is needed [7].
Most importantly, QUIC offers native support for multiplexing multiple data streams over a single connection. These streams represent bidirectional byte-streams, and each comes with its own priority, flow control, and congestion control. Re-transmission also occurs on a per-stream basis so that dropped packets in a stream do not inhibit the performance of other streams sharing the same connection, effectively solving the Head-of-Line blocking problem outlined in Section 2.3.1.

2.3. The Current State of Tor

Tor has long suffered from serious performance problems. Some of the notable causes include ineffective congestion control, unfair path selection, and low network capacity [2]. And while a lot of research efforts have gone into addressing these issues, many of them proved either ineffective or raised security concerns, proving the need for further research on the matter [3]. In this subsection, we will go over two particular problems affecting Tor's performance that can be potentially solved using QUIC.

2.3.1. Head-Of-Line Blocking. Head-Of-Line (HoL) blocking is a problem that occurs because of TCP's in-order property, which ensures that data is received in the same order it was sent in. When a TCP segment is lost, all succeeding packets must wait for successful re-transmission. Since multiple Tor circuits are multiplexed over a single TCP connection, this puts a halt to data transmission over all circuits, even if they are unrelated to the lost segment. As shown in Figure 2, if a high-bandwidth, high-latency stream *(e.g. bulk transfer)* shares a TCP connection with an unrelated low-bandwidth, low-latency stream *(e.g. web browsing)*, the latter could experience throttling, affecting the fairness of the network [2]. Basyoni *et al.* mention in [8] that as the popularity of Tor increases, it is expected that this will occur more frequently.
Nowlan *et al.* [9] attempted to address this with uTor: a Tor implementation that used un-ordered TCP to ensure that data is transmitted despite lost packets. However, this approach showed "modest performance gains", likely because of the additional processing of packets it required [2].

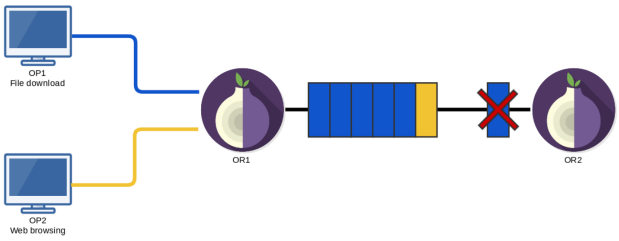


Figure 2: The Head-of-Line blocking problem in Tor. The blocks represent packets, their color corresponding to the originating client. Since one of *OP1*'s packets is lost, all other packets in transit are blocked until successful re-transmission, including *OP2*'s unrelated packet. Taken from [5].

2.3.2. Congestion Control. Ineffective congestion control is a major contributor to Tor's performance issues. Hop-by-hop congestion is managed by the TCP connections between nodes. However, due to Tor multiplexing multiple circuits over a single TCP connection, any slowdown caused by congestion will affect all the circuits over the connection, causing the same effects discussed in the previous section (2.3.1).
More importantly, end-to-end congestion control is simply ineffective. Unlike circuit level congestion control, there is no end-to-end TCP connection between the client and the exit relay. Because Tor connections are composed of multiple independent TCP connections, Tor does not currently have a "low latency method of informing the client of congestion in later links of the circuit" [10]. When a Tor relay receives a cell, it is required to forward it reliably. This makes it unable to drop the cell and causes its buffer to fill up when it receives more than it can send. Combined with other scheduling issues [11], [12], this results in relays being overburdened, with no effective way of informing the client to decrease its sending rate [10]. Currently, Tor uses a sliding window mechanism for end-to-end congestion control, but it has not been effective at reducing latency [13].

3. Proposed Designs for Tor over QUIC

In this section, we will go over the current proposed designs for Tor over QUIC that were briefly mentioned in the previous section. Most of the research community has adopted the hop-by-hop design, making use of QUIC's powerful streams to replace the existing TCP connections with QUIC ones [5], [8], [10], [14]–[16]. On the other hand, members of the Tor community have proposed an alternative end-to-end design that aims to solve Tor's congestion control problems [17]. In this section, we will examine these designs, their purposes, and their drawbacks.

3.1. End-To-End

Members of the Tor community have taken a very different approach to the research community, strongly deviating from Tor's current design that was presented in Section 2.1. In this design [17], Tor's circuit is replaced by a single end-to-end QUIC connection between the client

 doi: 10.2313/NET-2023-11-1_14

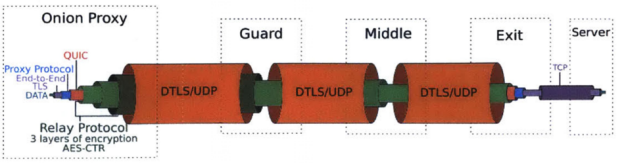


Figure 3: End-To-End Design. Taken from [10].

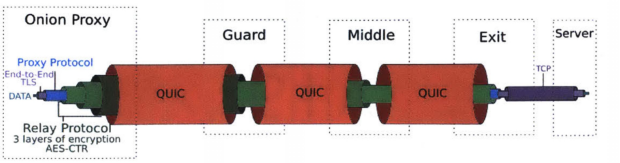


Figure 4: Hop-By-Hop Design. Taken from [10].

and the exit node and the TCP connections between Onion Routers are replaced with DTLS connections. DTLS is a protocol based on TLS that aims to secure datagram-based communication, such as UDP traffic [18]. This enables an unreliable and faster encrypted connection between relays, leaving congestion control to the inner traffic layers. The client and the exit node are then able to establish a QUIC connection by sending QUIC packets packaged into onion-encrypted Tor cells through the circuit (Figure 3). At the end of the circuit, the exit node then converts the connection back to TCP to communicate with the server. This design aims to solve Tor's inefficient congestion control by adding end-to-end congestion feedback between the client and the exit node, making use of QUIC's powerful and flexible congestion control.
Research has proven that congestion and other queuing-related problems are significant contributors to Tor's latency, even more than HoL blocking [11], [12], [19]. But while this design may have the greatest potential to improve Tor's performance, it comes with serious drawbacks. Kyle H. [10] points out several major flaws within this design. First, it completely changes Tor's original design, making it difficult to implement and deploy. This means that it could compromise Tor's security by causing unintended side effects. In fact, Sy *et al.* [20] show that end-to-end connections over QUIC facilitate web tracking, potentially compromising client anonymity. On the other hand, Tscorsch *et al.* [19] showed that end-to-end paths with high round trip times can impact fairness and increase latency due to longer reaction times.

3.2. Hop-By-Hop

Most existing proposals from the research community follow the same hop-by-hop design, staying faithful to Tor's current protocol [5], [8], [10], [14]–[16]. In this design, Tor's existing TCP connections are simply replaced by QUIC connections, and different circuits are multiplexed over different QUIC-streams within the same connection. Note that the connection between the exit node and the server has been kept as it is, as shown in Figure 4. The reason for this is that many servers do not support QUIC yet. In fact, QUIC traffic only represented 7% of all internet traffic in 2017 [21].
This approach has two major advantages: It immediately solves the HoL blocking problem because QUIC provides several logical streams over a single connection, making it so that loss of data in a stream does not affect the others. Furthermore, QUIC comes with a pluggable congestion control module that can be configured separately for each stream. This can be used to improve fairness between multiple Tor circuits, as congestion on one connection no longer affects all streams that use that connection.
However, it is important to note that this does not solve most of Tor's congestion control problems that were discussed in Section 2.3.2. In fact, most implementations of this design completely ignore end-to-end congestion and do nothing to address it. While the end-to-end design has its fair share of problems, it can still serve as inspiration on how to fix Tor's lack of end-to-end congestion control. Kyle H. [10] was the first to consider the potential for backpressure-based congestion control using QUIC. Essentially, this was achieved using QUIC's per-stream flow control, by allowing Onion Routers that experience congestion caused by a specific circuit to reduce the maximum data it accepts from the stream associated with that circuit. This, in turn, forces the previous relay to reduce its sending rate, potentially propagating this information back until the client's sending rate is reduced.

4. Results

Ideally, a Tor over QUIC implementation would solve the HoL problem, improve fairness in the network, improve congestion control, and ultimately not compromise Tor's security. In this section, we will proceed to examine whether these design goals have been successfully achieved or not, from both a performance and security standpoint.
Note that we will only examine the hop-by-hop design, as no implementations currently exist for the end-to-end one.

4.1. Performance

In this subsection, we will focus on the performance evaluation of the Tor over QUIC implementations proposed by Basyoni *et al.* and J. Heijligers [8], [16], as they are the most recent and come with detailed performance evaluations.
There are 2 important metrics that are used to evaluate the performance of a Tor implementation. *Time to First Byte* (TTFB) represents the time it takes for the client to establish a circuit and receive its first byte. Similarly, *Time to Last Byte* (TTLB) represents the time it takes for the client to establish a circuit and receive the last byte from the server.
All performance measurements outlined in the section follow the model laid out by Jansen *et al.* in [22] for accurate performance evaluations. Specifically, there were two types of simulated clients: low-bandwidth clients sending regular HTTP requests to represent web browsing and high-bandwidth clients performing bulk downloads (*e.g.* over BitTorrent).
QuicTor [8] reported very minimal improvement regarding TTFB for low-bandwidth clients. However, it managed to

 doi: 10.2313/NET-2023-11-1_14

reduce the TTLB for high-bandwidth clients by almost 80%, likely because these clients are more prone to being affected by HoL blocking. It also reported significant improvements for video streaming applications. Overall, it managed to outperform vanilla Tor in all scenarios and even mostly outperformed two other Tor implementations designed to address the problem of circuit multiplexing [23], [24].
Similarly, J. Heijligers [16] reported a 50% performance improvement over vanilla Tor. It also showed more fairness in distributing bandwidth among clients, reaching a near-perfect score on Jain's fairness index [25]. Although these measurements may not be perfect and do not exactly simulate real-time conditions, they show very promising results. It's also worth mentioning that these improvements are mostly attributed to improved circuit-level congestion control, since the hop-by-hop design does not address end-to-end congestion control.

4.2. Security and Privacy

Security and Privacy are fundamental aspects of Tor, as client anonymity is the major goal of the network. It is therefore important that Tor over QUIC implementations do not compromise security, either through exposing new attack vectors, aggravating existing ones, or undermining Tor's current defense mechanisms. In this subsection, we will focus on examining [8] and [10], as they include a comprehensive security analysis.
QuicTor [8] investigated the impact of switching to QUIC on *traffic correlation* attacks, as they are more likely to be impacted by a change in the transport layer protocol. In such an attack, an adversary would try to correlate traffic observed at one of Tor's nodes and one of its endpoints to deanonymize the client. The authors implemented two known attacks [26], [27] and observed that QuicTor did not behave much differently than vanilla Tor, even showing better resistance in some cases. This is due to timing-based attacks becoming less effective because of QUIC since they assume that one stream can influence all other streams passing through the same node [26].
Kyle H. [10] provided a more in-depth security analysis of his proposal by investigating whether QUIC leaks any extra information that may be used for an attack vector. The author concluded that it was important not to use more than a QUIC-stream per Tor-circuit, as per-object streams could leak detailed information about client traffic. 0-RTT connections also come with some security concerns, as they can allow client-tracking across different sessions, but countermeasures against this exist [20].
Furthermore, the author investigated whether his proposal for backpressure-based congestion control comes with any security risks, and concluded that these could be mitigated by only backpropagating limited information at set intervals.
However, some concerns remain about *website fingerprinting* (WF) attacks. These aim to analyze traffic to conclude which website the client visited. Nie *et al.* [28] developed QUIC-CNN, a novel model for WF attacks on QUIC traffic on Tor, and even found that it performs better than the current state-of-the-art model.

5. Conclusion and Future Work

In this work, we presented a literature review over the current state of Tor over QUIC. Tor currently has significant performance problems, and it is the main motive behind this change in its transport layer protocol to solve them without compromising security.
We examined two proposed designs: an end-to-end design that tunnels a QUIC connection between the client and the exit node, aiming to solve Tor's congestion related issues, and a hop-By-hop design that replaces Tor's current TCP connections with QUIC ones, aiming to solve the Head-of-Line blocking problem and improve fairness. Although the former is riddled with performance and security concerns, an ideal Tor over QUIC implementation should still strive for its goal of improving end-to-end flow control. We presented one design proposal [10] that suggested doing this through backpropagation of flow control information. While performance tests have been promising so far, more work still needs to be done on the matter. The aforementioned design proposal has still not been implemented, and its claims about improving performance still need to be corroborated. Furthermore, the security behind these designs still needs to be studied, as current research only provided a purely theoretical analysis or only considered one specific type of attack vectors.
Although these designs seem promising, we still have a long way to go before Tor finally switches over to QUIC.

References

[1] "Welcome to Tor Metrics," https://metrics.torproject.org, [Accessed 30-May-2023].

[2] S. J. M. Roger Dingledine, "Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it," *Tor Tech Report 2009-11-001*, [Online; accessed 15/05/2023; https://research.torproject.org/techreports/performance-2009-11-09.pdf].

[3] I. G. Mashael AlSabah, "Performance and Security Improvements for Tor: A Survey," *ACM Comput. Surv.*, vol. 49, no. 2, 2016.

[4] R. Dingledine and N. Mathewson, "Tor Protocol Specification," (visited on 28-05-2023). [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt

[5] W. Sabée, "Adding QUIC support to the Tor network," 2019.

[6] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. B. Krasic, C. Shi, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. C. Dorfman, J. Roskind, J. Kulik, P. G. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, and W.-T. Chang, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," 2017.

[7] M. Fischlin and F. Günther, "Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 60–75.

[8] L. Basyoni, A. Erbad, M. Alsabah, N. Fetais, A. Mohamed, and M. Guizani, "QuicTor: Enhancing Tor for Real-Time Communication Using QUIC Transport Protocol," *IEEE Access*, vol. 9, pp. 28 769–28 784, 2021.

[9] M. F. Nowlan, D. I. Wolinsky, and B. Ford, "Reducing Latency in Tor Circuits with Unordered Delivery," in *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*. Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/nowlan

[10] K. Hogan, "Security analysis of Tor over QUIC," 2020.

[11] Rob Jansen and John Geddes and Chris Wacek and Micah Sherr and Paul Syverson, "Never been KIST: Tor's congestion management blossoms with Kernel-Informed socket transport," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 127–142. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/jansen

[12] R. Jansen and M. Traudt, "Tor's Been KIST: A Case Study of Transitioning Tor Research to Practice," 2017.

[13] F. Tschorsch and B. Scheuermann, "Mind the gap: towards a backpressure-based transport protocol for the Tor network," *Networked Systems Design and Implementation*, Mar 2016.

[14] R. Aissaoui, O. Erdene-Ochir, M. Al-Sabah, and A. Erbad, "QUTor: QUIC-based Transport Architecture for Anonymous Communication Overlay Networks," in *Qatar Foundation Annual Research Conference Proceedings Volume 2016 Issue 1*, vol. 2016, no. 1. Hamad bin Khalifa University Press (HBKU Press), 2016, p. ICTPP2961.

[15] A. Clark, "Quux: a quic un-multiplexing of the tor relay transport," 2016.

[16] J. Heijligers, "Tor over QUIC," 2021.

[17] M. Perry, "[tor-dev] The case for Tor-over-QUIC," [Online; accessed 22/05/2023; https://lists.torproject.org/pipermail/tor-dev/2018-March/013026.html].

[18] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[19] F. Tschorsch and B. Scheuermann, "How (not) to build a transport layer for anonymity overlays," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 101–106, 2013.

[20] Sy, Erik and Burkert, Christian and Federrath, Hannes and Fischer, Mathias, "A quic look at web tracking." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 255–266, 2019.

[21] 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*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[22] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine, "Methodically Modeling the Tor Network," in *5th Workshop on Cyber Security Experimentation and Test (CSET 12)*. Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/cset12/workshop-program/presentation/Jansen

[23] M. AlSabah and I. Goldberg, "PCTCP: Per-circuit TCP-over-IPsec transport for anonymous communication overlay networks," 11 2013, pp. 349–360.

[24] J. Geddes, R. Jansen, and N. Hopper, "IMUX: Managing Tor Connections from Two to Infinity, and Beyond," 11 2014, pp. 181–190.

[25] R. Jain, D. M. Chiu, and H. WR, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," *CoRR*, vol. cs.NI/9809099, 01 1998.

[26] S. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, 2005, pp. 183–195.

[27] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011, pp. 215–226.

[28] M. Nie, F. Zou, Y. Qin, T. Zheng, and Y. Wu, "QUIC-CNN: Website Fingerprinting for QUIC Traffic in Tor Network," in *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, 2022, pp. 663–671.

Structure and Origin of CT Based Domain Lists

Lorenz Lehle, Patrick Sattler*, Johannes Zirngibl*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: lorenz.lehle@tum.de, sattler@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—The participation of Certificate Authorities in Certificate Transparency has the side effect of publicly accessible certificates from which domain lists can be derived. In this paper we analyse such domain lists and lay special focus on wildcard domains because of their ability to prevent information leakage. We find that of all domains in the domain list 18.0 % are wildcard domains, and 8.6 % are wildcard domains at the first level. Most of these immediate wildcard domains appear in a certificate with the corresponding eSLD and no other domain. Furthermore we find several patterns in the distribution of domains, such as more eSLD with an even rather than an odd number of subdomains. Additionally we find that the leftmost labels of domains correlate with used services and reveal more information about the domain holders internal structure.**

***Index Terms*—domain lists, certificate transparency logs, internet scans**

1. Introduction

Domain lists are an integral part of many areas of research regarding network architectures. Researchers conduct Internet measurements to obtain information about the distribution of IP addresses, reachable hosts, and used protocols or services. Conventionally such scans require iterating through the whole IP address range, which is impossible for IPv6 due to the vast and largely unused address space. This is where domain lists prove useful. They contain domains of actually existing and used services, thereby alleviating the necessity to scan all addresses.

An alternative to these conventional domain lists are Certificate Transparency (CT) based domain lists. These are obtained by extracting the domain names noted in certificates issued by Certificate Authorities (CAs). In this paper we present an overview of Certificate Transparency and how CT based domain lists are created. Furthermore we analyse the structure and labels of domains that are contained in domain lists available to the *Chair of Network Architectures and Services*.

Firstly we present the necessary background about Certificate Transparency and introduce the terminology used for domain names. In Section 3 we present studies that examine the applicability of CT log based domain lists and security implications of CT logs. In Section 4 we analyse the structure and distribution of domains contained in a CT log based domain list. For this purpose we explore which domain labels are used in practice and how wildcards affect the obtainable information. Lastly we conclude our findings and discuss how these domain lists can be used in further research.

2. Background

This section explains the details of Certificate Transparency, outlines terminology used later in the analysis of the domains and explains the importance of wildcard domains in the context of this paper.

2.1. Certificate Transparency

Certificates are the basis of confidential and authentic communication in the modern Internet and are issued by Certificate Authorities. The obtained authenticity is based on the chain of trust which reaches from a root Certificate Authority over intermediate CAs to issued certificates. If a client trusts a root CA, it implicitly trusts all certificates issued by intermediate CA. Therefore authenticity is based solely on trust into one or only few entities without further means of verification. If adversaries gain access to key material of a CA, it is possible that illegitimate certificates are issued to entities which are not the owners of the respective domains. An instance of such a misissuance occurred in 2011 where a breach at DigiNotar resulted in fraudulently issued certificates. [1]

To combat misissuance or malpractice of CAs, Certificate Transparency was introduced by Google following the DigiNotar misissuance incident. It is defined in RFC 6962 [2] and provides a way to monitor CAs and the certificates they produce. CT logs are append-only data structures that are operated by CAs or independent organisations like Google. A participating CA appends issued certificates to one or multiple CT logs. These CT logs can then be audited by domain owners or independent monitors. This way CAs can be held accountable and misissued certificates can be detected faster [3]. Modern browsers like Google Chrome only accept certificates for web traffic if they have been recorded in at least one CT log [4]. Additionally Chrome enforces a maximum validity duration of 398 days for certificates [5]. This creates an incentive for service providers to regularly renew certificates and for CAs to append them to CT logs.

While monitoring for misissuance is the primary purpose of CT logs, these public logs can also be parsed to obtain a list of domains for which certificates have been issued. This is done by extracting the *Common Name* (CN) and *Subject Alternative Names* (SAN) from a certificate which specify the *Fully Qualified Domain Names* (FQDNs) the certificate is valid for [6].

 doi: 10.2313/NET-2023-11-1_15

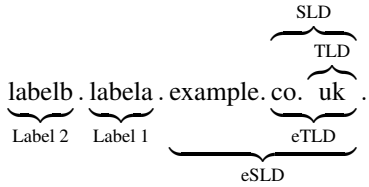


Figure 1: Structure of an FQDN with distinction between conventional TLD and SLD, and the introduced effective TLD (eTLD) and effective SLD (eSLD) with sub labels

2.2. DNS Terminology

According to RFC 8499 [7] a domain name consists of one or more labels that are separated by dots. Conventionally, the rightmost label is the Top Level Domain (TLD) and the label to its left in combination with the TLD is called the Second Level Domain (SLD) (see Figure 1).

For the analysis, we split a domain into a public suffix and private user-controlled part. The public suffix is in most cases equivalent to a TLD, but can also be a SLD because some registries use fixed SLDs for certain purposes. Examples for such domains are `co.uk` and `com.au`. The public suffix list [8] is a comprehensive list of these public suffixes. We call the public suffix the effective TLD and define the effective SLD, which is also called the private suffix, analogously to the SLD. Any further sub labels are the first, second, third, ... label of a domain.

2.3. Wildcard Domains

The domain name system allows the creation of records for wildcard domains. RFC 4592 [9] states that wildcard domains are domains where the leftmost label is an asterisk (`*`). A domain matches a wildcard domain when all labels of the domain, except for the label where the the asterisk is located in the wildcard domain, are identical. This specifically means that the domain `example.com` and domains with more labels like `a.b.example.com` do not match `*.example.com`.

Wildcard domains are especially relevant in the context of CT based domain lists, because they can hide information about operated services. If individual certificates are created for all subdomains (i.e. services) of an organisation, they are logged in a CT log. This unintentionally publishes identifiers that can be used to estimate the type of operated services, assuming that sensible names are chosen for the subdomains. When creating a certificate for the wildcard domain instead, the subdomains are not published.

3. Related Work

With certificate transparency being a relatively new component of the Internet, it has just been picked up by research in recent years.

Marquardt and Schmidt [10] examined whether CT based domain lists can be a viable alternative to other common domain top lists such as the Majestic or the, now discontinued, Alexa domain lists. Their reasoning for the search of alternatives was that the acquiration of these top lists is often not clearly defined and that CT based domain lists may be a well defined alternative. They used the FQDNs obtained from logged certificates and performed active measurements to compare the created list against the conventional domain lists. They found that while there are 30 % to 50 % less responsive hosts and in general more errors in name resolution with the CT based domain list, such lists can be used as a supplement for the conventional domain lists.

Scheitle et al. [11] examined the implications on security and privacy that arise when certificates are logged in CT logs. The use of CT has the consequence that FQDNs and therefore information about the structure of services is publicly logged. They performed a static analysis of domains to find labels of commonly used services. However they did not consider the depth of the label. They have also set up a *CT honeypot* to see whether CT logs are monitored by active parties. The honeypot hosts were reachable under random domain names which were only published in CT logs. They found that the domains were queried shortly after the certificates were published in a CT log by both presumably well intentioned services like Google or DigitalOcean, but also by suspicious sources. They conclude that CT does remedy one attack vector of certificate misissuance, but argue, that it may introduce new attack vectors based on the suspicious queries.

Pletinckx et al. improved this honeypot experiment in [12]. They ran the experiment for a longer time, used a larger number of hosts, and had a control group where the hosts had self signed certificates installed that were not submitted to any CT log. They noticed that the hosts with the logged certificates received significantly more traffic than the hosts with the unrecorded certificate, especially immediately after the certificate was issued, thereby confirming the previous work of *Scheitle et al.* They performed these experiments with both IPv4 and IPv6 hosts and the IPv6 hosts with self signed certificates experienced no traffic at all. This is explained by the mostly unused IPv6 address space, which, compared to the IPv4 address space, cannot be scanned on a regular basis. Third parties are therefore bound to rely on information like domain names obtained from CT logs to facilitate scanning in the IPv6 address space.

4. Analysis

In this section we analyse domain lists that were extracted from CT logs as described in Section 2.1. The domain lists are available on a per day basis since August of 2022 and contain only unique domains. In the analysis we use domain lists that represent a full month. The month lists contain all unique domains retrieved from certificates issued on the different days of the month. Because the lists contain unique names, there is no bias through multiple certificates issued for the same domain. This analysis focuses on the distribution of registered domains across the available eSLDs and the structure of subdomains configured by the respective domain holders.

The data set used for this analysis is the domain list retrieved in March 2023. We conducted the same analysis shown below for the neighbouring months of January and February and obtained similar results. Therefore we consider one month a representative time frame.

 doi: 10.2313/NET-2023-11-1_15

4.1. Number of Domains

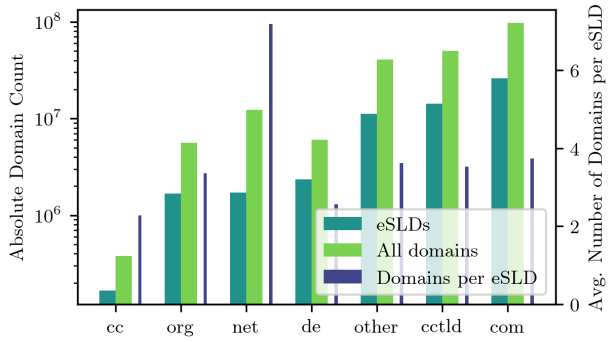


Figure 2: Absolute distribution of both unique eSLDs and all domains across various eTLDs.

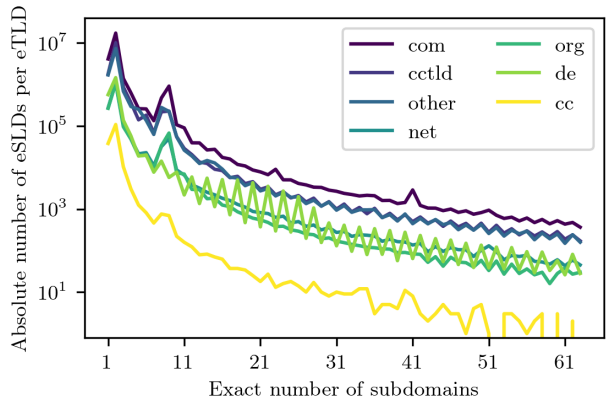


Figure 3: Number of exactly n subdomains per eSLD and eTLD

We begin the analysis by inspecting how many domains are in the data set and how they are distributed across eTLDs and eSLDs. The public suffix list [8] used here contains roughly 6800 unique public suffixes. Because of this large number of suffixes we limit our analysis to the prevalent generic Top Level Domains (gTLDs) `com`, `net`, and `org` and selected Country Code Top Level Domains (ccTLDs) which are `de` as a regular country code domain and `cc`. We chose the latter one because it is a popular open ccTLD that can be registered by anyone. These openTLDs are often used as replacement for the more conventional TLDs like `com` [13]. All other ccTLDs are grouped and represented with `ccTLD`; all remaining public suffixes are represented with `other`.

In Figure 2 we see that most of the domains read from the CT log are under the `com` eTLD. The number of eSLDs under `org` and `net` is by a factor of 10 to 15 less than of the `com` eTLD and in the same magnitude as the country code domain `de`. We observe that the single `com` eTLD has more eSLDs and domains than all country code TLDs or all gTLDs combined. In addition to the absolute number of domains, the graph shows the average number of domains under a single eSLD per eTLD. Here we see that the `net` eTLD has the most logged domains per eSLD with an average of 7.2, while the eSLDs in other eTLDs have an average of 2 to 3 domains.

Figure 3 shows the number of eSLDs in an eTLD that have exactly a specific number of domains. This includes the eSLD itself, conventional subdomains, and wildcard subdomains. Here we see an exponential drop, where most eSLDs do not have more than 10 domains. There are three noticeable deviations from the curve:

- In all cases there are slightly more eSLDs with exactly two domains than with exactly one domain.
- There is a burst of domains at 8 and 9 domains. An inspection of the eTLDs and eSLDs where 8 and 9 domains were present did not yield any pattern that would explain this phenomenon.
- Additionally there is a repeating zig-zag pattern that indicates that there are more eSLDs that have an even number of domains rather than an odd number. This pattern begins to emerge at a number of about 10 domains per eSLD and is especially pronounced in the `de` eTLD.

4.2. Wildcard Certificates

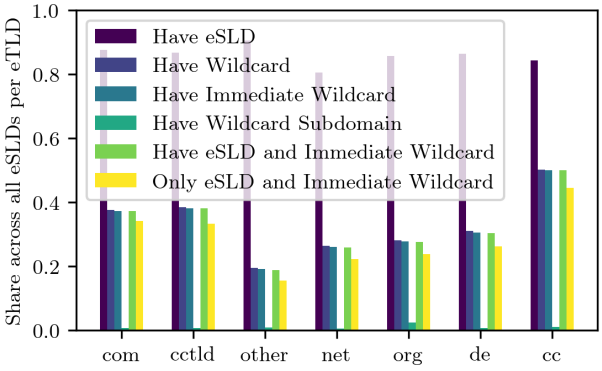


Figure 4: Relative amount of eSLDs per eTLD that fulfil the corresponding criterion

We pay special attention to wildcard domains, because of their ability to hide information as explained in Section 2.3. In this section we analyse how wildcard domains are distributed and used.

We label eSLDs depending on whether the eSLD itself, any wildcard domain, the immediate wildcard domain (in the form `*.eSLD`), or a wildcard subdomain are contained in the domain list. Figure 4 shows that, with a share of 80.5 % to 90.5 %, nearly all eSLDs have a certificate for the eSLD itself and about 19 % to 50 % have one for the wildcard domain, which is in most cases the immediate wildcard. There are very little eSLDs that have a certificate for a wildcard subdomain in the form `*.label.eSLD`. The most significant set of domains are these where there is a certificate for both the eSLD and the immediate wildcard: The number of eSLDs where this is the case is only marginally higher than the number of domains where *only* these domains had a certificate. In total 10.5 % of domains were immediate wildcard domains.

This phenomenon generalises across all eTLDs and has an apparent reason. It is a widely adopted use case to obtain a certificate for both the eSLD and the immediate wildcard domain and no other domains. This approach is suitable if there is no need for subdomains with more than

 doi: 10.2313/NET-2023-11-1_15

one label and makes the deployment of new services under additional first level subdomains very simple because no new certificates have to be generated and the wildcard certificate is sufficient. However, this does hide the internal structure and information about existing hosts reachable under that domain. While this is a privacy improvement, it hinders the usage of CT based domain lists for host reconnaissance.

4.3. Leftmost Label

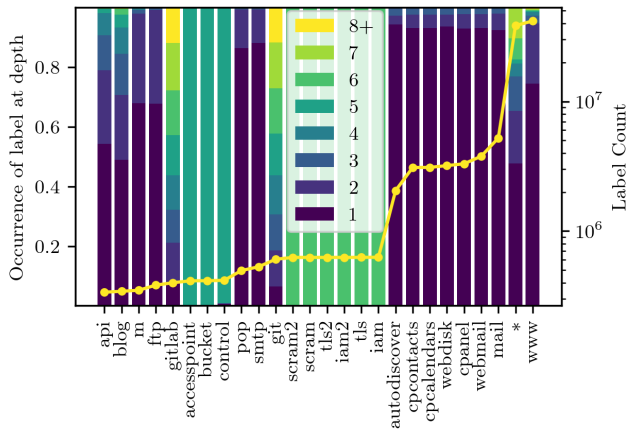


Figure 5: Distribution of most prevalent leftmost labels across different depths in all read domains including the absolute occurrence count of the respective label

The leftmost label of a domain allows for an estimation of the service available on the target host. Figure 5 shows the leftmost labels that were most prevalent across all evaluated domains along with their depth distribution. Labels that were found at a depth of 8 or more are grouped into a single section, however there are almost no popular labels at a depth higher than 7.

We distinguish three different types of distributions that emerge in the most popular labels:

- Distribution that decreases exponentially with increasing depth as seen with, among others, the labels `*` (wildcard), `www`, and `blog` and also less pronounced with e.g. `mail` and `cpanel`.
- Occurrence almost exclusively at a certain depth as seen with, among others, the `iam`, `tls`, and `bucket` labels.
- Mostly equal distribution across all depths as seen with the `git` and `gitlab` labels.

4.3.1. Exponential Decrease. The exponentially decreasing distribution is what one would normally expect for user facing domains. Domains with more labels are not that common for this use case and instead mostly used in the context of deployments or deep internal hierarchies of large organisations. Conventionally `www` is used for web services and makes up for 19.7 % of all read domains alone. Compared to this we see that the wildcard label `*` occurs equally as often as the `www` label with 18.0 %. Combined with the results from Section 4.2, this means that a large number of wildcard labels at deeper levels are concentrated towards a low number of eSLDs.

The labels `mail` or `webmail` hint at the use for a web interface to manage an email inbox. Other labels like `cpanel`, `webdisk`, `cpanelcalendars`, and `cpanelcontacts` also allow for a specific service estimation. These labels are commonly used by instances of the web hosting management software *cpanel* [14]. We see that at least 95 % of user targeting domains such as `www`, `mail` or `cpanel` are only one or two labels long. Of all read domains 30.2 % we such user facing domains.

4.3.2. Equal Distribution. The equal distribution across the depths 1 through 7 is only present with `git` and `gitlab`. There is a pattern in the data set, where the labels `git` and `gitlab` are permuted for up to 6 labels like `(gitlab|git).(gitlab|git)....` There is no apparent correlation to any eTLD or eSLD in the domains with that behaviour. In fact the distribution across the public suffixes was comparable to the distribution of all domains.

4.3.3. Single Depth Labels. The labels `iam`, `tls`, and `scram` and the duplicates with suffix `2` appear only at depth 6 and the labels `bucket` and `accesspoint` similarly only appear at depth 5. This differs from labels such as `cpanel` which, even though they occur at multiple depths, predominantly occur at depth 1. An inspection of the domains with these labels showed that the corresponding eSLD is `amazonaws.com` in 95 % of all cases for `bucket` and `accesspoint` and in 92 % of all cases for `iam`, `tls` and `scram`. The correlation with amazon is backed up by the fact that amazon uses the term *bucket* for its cloud object storage S3 web service [15] and *iam* for *Identity and Access Management* [16]. In these cases, the intermediate labels of the domain encode location information. These domains related to Amazon cover a total of 2.44 % of all domains in the used domain list.

5. Conclusion and Future Work

From our analysis we can see that the `com` eTLD is still the most prevalent one, despite the rising number of other gTLDs. We also find patterns like an accumulation at 8 and 9 subdomains or the preference for an even number of domains in the distribution of subdomains. Further research might clarify the underlying cause of these patterns. The evaluation of wildcard domains yields that certificates are widely used in the common configuration where the eSLD and the immediate wildcard domain are covered by the certificate. Inspecting the leftmost labels of the domains also makes it possible to identify the associated services as we have seen with the *Amazon* and *cpanel* services.

The combination of these findings might make it possible to work around the information hiding ability of wildcard domains. Future work may experiment with substituting the wildcard label with other common and concrete leftmost labels from non-wildcard domains. This could yield further host names that resolve and complete a CT based domain list to make it more comparable to other domain lists.

 doi: 10.2313/NET-2023-11-1_15

References

[1] H. Hoogstraaten, "Black Tulip Report of the Investigation into the DigiNotar Certificate Authority Breach," August 2012.

[2] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6962

[3] O. Gasser, B. Hof, M. Helm, M. Korczynski, R. Holz, and G. Carle, "In Log We Trust: Revealing Poor Security Practices with Certificate Transparency Logs and Internet Measurements," in *Passive and Active Measurement*, R. Beverly, G. Smaragdakis, and A. Feldmann, Eds. Cham: Springer International Publishing, 2018, pp. 173–185.

[4] Chromium, "Google Chrome Certificate Transparency Policy." [Online]. Available: https://github.com/GoogleChrome/CertificateTransparency/blob/master/ct_policy.md

[5] ——, "Google Chrome Certificate Lifetimes." [Online]. Available: https://github.com/chromium/chromium/blob/main/net/docs/certificate_lifetimes.md

[6] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5280

[7] P. E. Hoffman, A. Sullivan, and K. Fujiwara, "DNS Terminology," RFC 8499, Jan. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8499

[8] Mozilla Foundation, "Public Suffix List," 2022. [Online]. Available: https://publicsuffix.org/

[9] E. P. Lewis, "The Role of Wildcards in the Domain Name System," RFC 4592, Jul. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4592

[10] F. Marquardt and C. Schmidt, "Don't Stop at the Top: Using Certificate Transparency Logs to Extend Domain Lists for Web Security Studies," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 409–412.

[11] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, "The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem," ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 343–349. [Online]. Available: https://doi.org/10.1145/3278532.3278562

[12] S. Pletinckx, T.-D. Nguyen, T. Fiebig, C. Kruegel, and G. Vigna, "Certifiably Vulnerable: Using Certificate Transparency Logs for Target Reconnaissance," 2005. [Online]. Available: https://hdl.handle.net/21.11116/0000-000C-F940-3

[13] "Country Code Top-Level Domain." [Online]. Available: https://icannwiki.org/Country_code_top-level_domain

[14] cPanel, L.L.C., "cPanel Products." [Online]. Available: https://www.cpanel.net/products/

[15] Amazon Web Services, Inc., "Amazon S3." [Online]. Available: https://aws.amazon.com/s3/

[16] ——, "Amazon IAM - AWS Identity and Access Management." [Online]. Available: https://aws.amazon.com/de/iam/

Introduction to BBRv2 Congestion Control

Joji Mathew, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: joji.mathew@tum.de, jaeger@net.in.tum.de

***Abstract*—Congestion happens when an overflow occurs due to more data being sent than the bandwidth and this leads to loss. Congestion control algorithms try to prevent this and the ones currently used widely are TCP algorithms which are loss-based and delay-based starting only once a loss has occurred leading to retransmissions due to which further congestion occurs. Google proposed BBR in 2016 using another approach called congestion-based congestion control which acts at an earlier stage compared to TCP algorithms promising maximum throughput and minimum queuing delay. But it still comes with drawbacks that Google is trying to solve with the second version BBRv2 which is still in the testing stage. In this paper, we give an overview on congestion control overall, BBR, and introduce BBRv2.**

***Index Terms*—congestion control, TCP, BBR, CUBIC, bottleneck bandwidth**

1. Introduction

TCP (Transmission Control Protocol) is a protocol in the transport layer that ensures to establish end to end connection between two hosts and is also responsible for the safe transmission of data between the two. Whenever a data packet is lost it makes sure to re-transmit it. When multiple senders, unaware of the bandwidth of the connection send lots of data, it leads to congestion in the network which causes buffer overflow and packet loss. TCP uses congestion control algorithms to avoid this situation [1]. TCP has several implementations for the same starting with RENO to the present-day CUBIC.

Excessive buffering and delays were reported on the Internet in 2011 [2]. This was mainly due to the fact that TCP operates only having loss as a parameter and starts operating too late once congestion has occurred. By this time senders re-transmit lost data which leads to further congestion. BBR which was developed by Google tries to operate at Kleinrock's optimal point [3] way before a loss occurs.

The paper is structured as follows. Section 2 talks about the different approaches used in congestion control. An introduction to BBR, it's working, and performance analysis is discussed in Section 3. Section 4 talks about some drawbacks of BBR. BBRv2 is introduced briefly and an elaboration of its algorithm and performance analysis is done in Section 5. Finally, the conclusion and future work is given in Section 6.

2. Different approaches in congestion control

These are some terms needed for later and their explanations according to Scholz et al. in [3].

- ***BtlBW***- Bottleneck is a link in the network path with the lowest bandwidth and this bandwidth is called Bottleneck bandwidth(BtlBW).
- ***RTT***- Round Trip Time(RTT) is the time required for a data packet to reach the receiver plus the time for the acknowledgment to reach the sender.
- ***RTprop***- It is the round trip propagation delay when there is no queuing delay and very less data in the buffer.
- ***Inflight data***- Amount of data in the buffer.
- ***BDP***- RTprop · BtlBW is called a pipe's Bandwidth-Delay Product(BDP).
- ***cwnd***- Amount of data being sent in each RTT

2.1. TCP congestion control algorithms

TCP has primarily two congestion control approaches. Even though it has a lot of algorithms in development. The following are the ones used widely:

Loss-based congestion control. This approach identifies loss as congestion and is used in implementations like RENO, CUBIC. The basic idea of the implementation is that it starts with a small cwnd, then increases exponentially after each RTT, and whenever a loss occurs, it interprets it as congestion and reduces the cwnd by a certain amount and now increases it unlike earlier only in small amounts. This process of reducing cwnd and slow incrementation keeps on repeating whenever a loss occurs [4].

Delay-based congestion control. This other approach identifies delay as a parameter for congestion instead of loss and starts working when it sees an increase in RTT.

2.2. Congestion based Congestion control

The problem with loss-based congestion is that it only starts acting when a loss occurs. When the Amount Inflight > BDP queue starts slowly building up in the buffer, increasing RTT, and when the Amount inflight= BDP + Bottleneck buffer size, the buffer becomes full, leading to congestion and packet loss [5].

In bigger buffers loss-based congestion control algorithm sends data at full bandwidth causing buffer bloat,

 doi: 10.2313/NET-2023-11-1_16

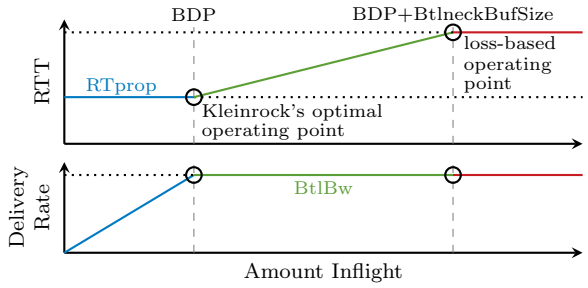


Figure 1: Amount Inflight vs Delivery Rate and RTT [3]

and only starts acting at BDP + Buffer size, where it is already too late since the loss of packets causes retransmission resulting in further delays [6]. In the older days, it used to make sense since memory was expensive and buffers were smaller so it minimized the delay of reaching BDP + Buffer size. But nowadays, with lesser prices and more memory available, it causes a delay of seconds. In the case of small buffers, this algorithm is also overly sensitive and bring about low throughput because it misinterprets loss as congestion, which could have occurred due to other reasons like flows entering the pipe.

The optimal point to start congestion control is when Amount Inflight = BDP, as shown in Figure 1, also known as Kleinrock's optimal point [6]. This condition ensures that the bottleneck does not starve, and at the same time, the data does not overfill the pipe. Also the bottleneck could be fully utilized if the sending and arrival rate of packets at the bottleneck are equal. BBR tries to satisfy both of these conditions.

3. BBR

BBR(Bottleneck Bandwidth and Roundtrip) was proposed as an alternative approach by Google in 2016 [3]. It is a congestion-based congestion control approach that mainly tries to achieve high throughput with a small queue and can withstand random losses upto $15\,\%$, according to Cardwell et al. in [7]. BBR operates on the parameter BDP mainly, and to measure BDP, it has to constantly determine BtlBW and RTT. But they cannot be observed simultaneously, since as shown in Figure 1, RTT can be measured accurately when Amount Inflight < BDP so that there is no congestion resulting in RTT = RTprop and BtlBW has to be measured when Amount Inflight > BDP so that we get the maximum bandwidth.

3.1. Algorithm

According to Scholz et al. in [3], BBR algorithm operates in 4 phases.

Phase 1(Startup): *Pacing gain* is a parameter that controls the amount of data sent, that when multiplied with the BtlBW shows the current sending rate. The sending rate is doubled after each roundtrip, and once the bandwidth has reached its maximum value, which BBR assumes as the BtlBW, it continues with the second phase.

Phase 2(Drain): Here, BBR tries to reduce the queue created at the bottleneck due to the first phase by temporarily decreasing the *pacing gain* and starts with next phase.

Phase 3(Probe Bandwidth): In this phase, with a total of 8 cycles BBR tries to estimate the BtlBW. In the first cycle, the *pacing gain* is set to 1.25 to probe for extra bandwidth and then at 0.75 to drain the queues created. Then for the rest of the phase, i.e., six cycles, the *pacing gain* is set to 1. The bandwidth is sampled constantly throughout the time and the maximum is used as the BtlBW. This value holds for a period of 10 RTprop. Then it enters the next phase.

Phase 4(Probe RTT): In this phase, which is about $200\,\mathrm{ms}$ plus one RTT, bandwidth is set to four packets to drain any possible queues created by the third phase to get the current estimation of RTT. RTprop value is updated if a new minimum is measured and is valid for tens seconds.Both Probe_BW and Probe_RTT phase are repeated continuously and updated.

3.2. Performance Analysis of BBR

When we look at the performance analysis of BBR and CUBIC done by Cardwell et al. in [7], it is to be seen that BBR achieves high bandwidth despite losses. At a loss rate of $0.01\,\%$, CUBIC lowers the throughput to around $30\,\mathrm{MB\,s^{-1}}$ and at $0.1\,\%$, further lowers to $12.5\,\mathrm{MB\,s^{-1}}$ while BBR maintains the maximum at around $90\,\mathrm{MB\,s^{-1}}$, which shows that even small losses lead to low throughput in a loss-based congestion control approach. When measuring the buffer size(MB) against latency (s), CUBIC has a linear increase in latency while BBR maintains a low queue delay despite bloated buffers.

4. Problems of BBR

Even though BBR promised to solve a lot, it still comes with problems. According to experiments done by Ma et al. in [8], Song et al. in [5], and Scholz et al. in [3], the following drawbacks were observed:

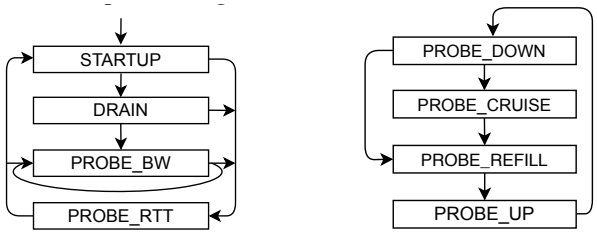
4.1. RTT unfairness

With multiple flows, flows with longer RTT recieve a larger share of bandwidth compared to flows with smaller RTT is the opposite of traditional loss-based and delay-based algorithms, which favor flows with shorter RTT. This leads to the following problems. There is an unpleasant trade-off between low latency and high delivery rates. It no longer makes any sense to find a route that has minimum RTT since flows with longer RTT receive more bandwidth leading to a high delivery rate. Secondly, since BBR is also a sender-based congestion control like TCP, it will be easier to manipulate and increase the RTT from the receiver side to get more share of bandwidth, and the sender would be totally unaware of the situation. This could lead to worse outcomes if all the receivers compete and try to inflate their RTT constantly.

4.2. Unresponsive to packet loss

BBR overestimates the bottleneck bandwidth and according to the analysis done by Hock et al. in [9], the

 doi: 10.2313/NET-2023-11-1_16



(a) Phases of BBRv1 & BBRv2 (b) PROBE_BW in BBRv2

Figure 2: Phases of BBR and BBRv2 algorithm [10]

Amount Inflight is between 2 BDP and 2.5 BDP, most of the time. This could lead to problems when the buffer size is less than 1 BDP since BDP + Buffersize, in this case,would be less than 2 BDP. Due to the overestimation BBR might start operating only at 2BDP, whereas loss has already started occurring at BDP + Buffersize and BBR would not reduce the amount of data sent since it does not respond to loss, leading to further retransmissions and delays.

4.3. Fairness between BBR and other algorithms

When BBR operates together with loss-based congestion like CUBIC, the share of the bandwidth varies according to the buffer size. When the buffer is shallow, BBR occupies more bandwidth and in the case of deep buffers CUBIC occupies a major share. In each of these cases, this behaviour leads to starvation of flows using the other algorithm leading to high retransmissions.

5. BBRv2

BBR was proposed by Google in 2016 [3] as an alternative to traditional loss and delay-based algorithms, promising maximum throughput with minimum queuing delay. To achieve this, BBR tries to work at Klienrock's optimal point. It was deployed in Google B4 WAN, Google.com and Youtube video servers [6]. B4 is a high-speed Wide Area Network from Google, and in B4, it was to be seen that BBR had a throughput 2 to 25 times greater than CUBIC, and when the receiver window was raised, BBR was 130 times faster than CUBIC. In Google.com, web page downloads were faster mainly in the developing world. On Youtube, playbacks using BBR had less rebuffering, and BBR was also able to achieve a median RTT of $53\,\%$ and $80\,\%$ in developing world.

Even though BBR showed promising results in the deployments, it still came with flaws. Apart from the ones discussed earlier, BBR also does not respond to ECN signals (Internet Protocol provides an extension called Explicit Congestion Notification(ECN) to notify TCP about network congestion without dropping the packet, thus avoiding retransmission [11]), and it has low throughput for paths with high aggregation(wifi). BBRv2, the newest version of BBR with modifications, was developed by Google in 2018 to solve these problems [12]. To coexist better with algorithms like CUBIC, it adapts bandwidth probing. BBRv2 also responds to loss and uses DCTCP-style ECN. Finally, it estimates the recent degree of aggregation to avoid low throughput problems [13].

	CUBIC	**BBR**	**BBRv2**
Parameters	None	BtlBW, RTT	BtlBW, RTT, max aggregation,max inflight
Response to loss	Yes	No	Yes
Response to ECN	Yes	No	Yes
Startup	slow until RTT rise or any loss	slow start until threshold	slow start until threshold or ECN/loss rate > target
cwnd in Probe RTT phase	N/A	4 packets	BDP/2

TABLE 1: Comparison table based on [12], [16]

A current draft for the BBRv2 is available on the internet [14] and as per the draft BBRv2 is currently available publicly for Linux TCP and QUIC. QUIC is a transport layer protocol like TCP implemented by Google and deployed in 2012. It is supported in major web browsers like Google Chrome, Firefox, and Microsoft Edge. It tries to establish a connection with less latency than TCP. QUIC also aims at bandwidth estimation in both directions, which in turn helps the BBRv2 at BtlBW estimations [15].

5.1. Algorithm

Explanation of the algorithm according to [14] and [10]. BBRv1 and BBRv2 algorithms work similarly in most cases. BBRv2 has some additional modifications on how it responds to loss and ECN. The Probe_Bandwidth phase is adaptive and therefore helps to coexist better with algorithms like CUBIC solving the interprotocol fairness problem. BBRv2 keeps tabs on the current estimate(.._latest), lower bound(.._lo), and higher bound(.._hi) values of the bandwidth and amount of data in flight. The congestion window is bounded by these lower and upper bound values instead of the inflight cap of 2 BDP, like in BBRv1. BBRv2 also has some minor changes with the congestion window in the drain phase.

Both BBR and BBRv2 exit the startup phase and enter the drain phase when there has been no significant increase of a minimum of $25\,\%$ in the bandwidth in the last three RTTs. In addition to this condition, BBRv2 also goes to the drain phase if there is packet loss or when ECN marks $50\,\%$ of the delivered packets for two consecutive RTTs.

The main difference in the algorithm is in the Probe_Bandwidth phase. According to Nandagiri et al. in [10], BBRv2 further breaks Probe Bandwidth into four sub-phases, as shown in Figure 2. In the Probe_Down phase BBR tries to reduce the amount of data in flight by lowering the sending rate to $90\,\%$ of the bottleneck

 doi: 10.2313/NET-2023-11-1_16

bandwidth. It exits this state when there is free headroom. So if the inflight_hi value is set, it remains in this state until the volume of inflight data is less than inflight_hi. The inflight_hi helps prevent loss and leaves space for other flows or cross traffic. Also, any queues created at the bottleneck have to be drained. Both of these conditions have to be met to exit this phase. In the Probe_Cruise phase, BBRv2 tries to send data at the same rate as the delivery rate, i.e., data is sent at 100 % capacity of bottleneck bandwidth. In this phase, it responds to loss and ECN by reducing bandwidth_lo and inflight_lo, indicating that the delivery rate and amount of data inflight need to be reduced. This phase holds adaptively and exits when it needs to probe for more bandwidth. In the Probe_Refill phase, BBRv2 has a goal of refilling the pipe. It attempts to send at 100 % of bottleneck bandwidth for just one more RTT, with bandwidth_hi and inflight_hi restraining the connection. In the Probe_Up phase, BBRv2 tries to probe for more bandwidth. The *pacing gain* is set to 1.25 which when multiplied by the bottleneck bandwidth, gives the current sending rate. When higher bandwidth and amount inflight values are measured, the bandwidth_hi and inflight_hi values are updated. It exits this phase when there is either a loss of 2 % or when the queue is high enough that the flow judges that it has probed adequately.

In the Probe RTT phase, which BBRv1 enters every 10 seconds to estimate the latest RTprop, BBRv1 sends only four packets to drain the queues created in Probe_BW whereas BBRv2 enters the phase every 5 seconds and maintains the throughput by draining it only to half the BDP in this phase.

Table 1, based on [16] and [12], shows the difference in how CUBIC, BBR, and BBRv2 work.

5.2. Performance analysis of BBR v2

Response to loss. As it can be seen in Figure 3 from Song et al. in [12], CUBIC is overly sensitive and starts reducing throughput from a loss of 0.001 %, BBR is loss agnostic and continues to deliver maximum throughput up to around 15 % and BBRv2 provides a good middle ground providing maximum throughput until about 1 % loss.

Inter-protocol Fairness. Experiments were conducted by Nandagiri et al. in [10] to check the Inter-protocol fairness. In the experiment, ECN was disabled since CUBIC and BBRv2 work with different types of ECN. In shallow buffers, BBRv1 had more share than CUBIC as pointed out earlier. But in the case of BBRv2, the sharing of bandwidth with CUBIC is fairer than BBRv1. This behaviour attributes to BBRv2 being bounded by lower and upper bound values and not having an inflight cap like BBRv1, preventing losses in the startup phase, helping CUBIC to maintain its throughput and not reduce the congestion window. After Startup both BBRv2 and CUBIC reduce throughput when they encounter loss. During the probing phase, the bandwidth of BBRv2 remains somewhat constant, but CUBIC on the contrary, increases its bandwidth after each RTT leading to having a slightly larger share.

In the case of deep buffers BBRv2 has a throughput way less than its fair share. Both CUBIC and BBRv2 have an equal throughput in the beginning and reduce

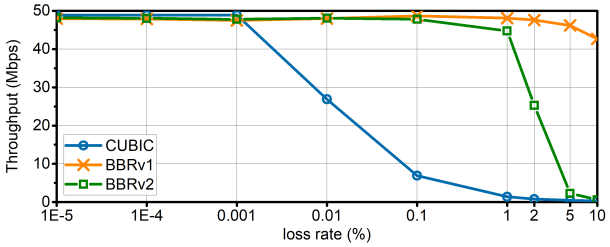


Figure 3: Throughput vs loss for CUBIC,BBR and BBRv2 [12]

throughput encountering a loss. But when CUBIC starts increasing its sending rate BBRv2 is still in the drain phase, trying to reduce the queues created at Startup phase. By the time BBRv2 enters the Probe_Bandwidth phase, it experiences loss due to CUBIC reaching the threshold forcing BBRv2 to reduce its bandwidth's higher bound further.

RTT fairness and Intra-protocol fairness. Experiments were conducted by Song et al. in [12] to determine the RTT and Intra-protocol fairness of BBRv2. It was done with two flows, with the first flow having a fixed RTT of 30 ms and the second flow with varying RTT. In BBRv1 flows with larger RTT received a bigger share of bandwidth than the shorter flows. In BBRv2 a similar behaviour is observed. For flows with the same RTT, the flow starting first gets a bigger share of throughput, but according to Nandagiri et al. in [10], similar flows have better fairness when ECN is enabled. Experiments on RTT fairness were also conducted by Gomez et al. in [17] with 50 flows of 10 ms RTT and 50 flows of 50 ms RTT. It is shown that both BBRv1 and BBRv2 allocate more bandwidth to flows with bigger RTT when buffer size is above 0.6 BDP, but when the buffer size is above 12 BDP, BBR v2 has a very high fairness index, whereas BBRv1 is still unfair.

6. Conclusion and future Work

BBRv2 is the best version of BBR, as of now. BBRv2 when deployed on Youtube had lesser RTT than CUBIC and BBRv1. It is currently deployed as the default TCP congestion control for internal Google traffic [16]. It provides maximum throughout till 1% loss, is more efficient using ECN signals, reduces queuing delay, and has better throughput in wifi. BBRv2 alpha is the current version which has some shortcomings which Google is working on fixing with the final version. BBRv2 has a fairness issue when ECN is disabled and is a bit too complex to deploy in WAN like b4 due to its dependency on DCTCP like ECN [10]. Although RTT fairness issues are better compared to BBRv1, it has not been solved completely. The same situation remains with inter-protocol fairness in the case of deep buffers. According to [18], Google is trying to add BBR.Swift extension to BBRv2. It aims to use delay also as a parameter without changing the BBR core. Google is also planning for a full-scale rollout in their company.

References

[1] GeeksforGeeks, "Transport Layer Responsibilities," [accessed 24-May-2023]. [Online]. Available: https://www.geeksforgeeks.org/transport-layer-responsibilities/

[2] Engagement and P. operations center, "20200504 - Neal Cardwell - BBR: A Model-based Congestion Control," https://www.youtube.com/watch?v=mpbWQbkl8_g, [accessed 17-July-2023].

[3] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of tcp bbr congestion control," in *2018 IFIP networking conference (IFIP networking) and workshops*. IEEE, 2018, pp. 1–9.

[4] GeeksforGeeks, "TCP Congestion control," [accessed 3-June-2023]. [Online]. Available: https://www.geeksforgeeks.org/tcp-congestion-control

[5] Y.-J. Song, G.-H. Kim, and Y.-Z. Cho, "Bbr-cws: improving the inter-protocol fairness of bbr," *Electronics*, vol. 9, no. 5, p. 862, 2020.

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

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev, and V. Jacobson, "Bbr congestion control: Ietf 99 update," in *Presentation in ICCRG at IETF 99th meeting*, 2017. [Online]. Available: https://www.ietf.org/proceedings/99/slides/slides-99-iccrg-iccrg-presentation-2-00.pdf

[8] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," *arXiv preprint arXiv:1706.09115*, 2017.

[9] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of bbr congestion control," in *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[10] A. Nandagiri, M. P. Tahiliani, V. Misra, and K. K. Ramakrishnan, "Bbrvl vs bbrv2: Examining performance differences through experimental evaluation," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, 2020, pp. 1–6.

[11] Wikipedia, "Explicit Congestion Notification," accessed 12-June-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Explicit_Congestion_Notification

[12] Y.-J. Song, G.-H. Kim, I. Mahmud, W.-K. Seo, and Y.-Z. Cho, "Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm," *IEEE Access*, vol. 9, pp. 37 131–37 145, 2021.

[13] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "Bbr v2 a model-based congestion control," 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

[14] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/

[15] Wikipedia, "QUIC," accessed 12-June-2023]. [Online]. Available: https://en.wikipedia.org/wiki/QUIC

[16] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao *et al.*, "Bbrv2: A model-based congestion control performance optimization," in *Proc. IETF 106th Meeting*, 2019, pp. 1–32. [Online]. Available: https://lafibre.info/testdebit/linux/201911_bbr_v2_doc_ietf106.pdf

[17] J. Gomez, E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "A performance evaluation of tcp bbrv2 alpha," in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, 2020, pp. 309–312.

[18] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, V. Jacobson, N. Dukkipati, and G. Kumar, "Bbr update : 1:bbr.swift; 2:scalable loss handling," 2020. [Online]. Available: https://datatracker.ietf.org/meeting/109/materials/slides-109-iccrg-update-on-bbrv2-00

The Evolution of Top-Level Domains: A Comparative Study of .org and .dev

Florian Pfisterer, Johannes Zirngibl*, Patrick Sattler*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: florian@pfisterer.dev, zirngibl@net.in.tum.de, sattler@net.in.tum.de

***Abstract*—The Domain Name System (DNS) is key to the modern Internet. Many studies investigate the changes of domain names over time. We argue that the the development of top level domains (TLDs) as a whole can introduce bias into such DNS measurements, leading researchers to draw conclusions about patterns in the data that actually stem from how the TLD changes over time. Additionally, one needs to take the churn of a given TLD into account when assessing the validity of DNS data given its age.**

To investigate how DNS data change varies across TLDs, we analyze the recent development of two very different TLDs between 2019 and 2023. By studying the zone files, we compare the almost 40 year old .org TLD with the much newer .dev TLD first made available only in 2019.

We aggregate zone file data over time and show that .org and .dev differ in a variety of aspects – .org is almost 30 times larger, grows only about three percent as fast, and exhibits 29 % less churn. We introduce the idea of the "core" of a zone (the domains that are registered for the full investigated period) and find that .org's core is 14 % larger than .dev's core, suggesting that we can consider .org a more stable TLD.

***Index Terms*—top-level domains, DNS, CZDS, zone files**

1. Introduction

Centralized Zone Data Service (CZDS) [1] is an online portal provided by the Internet Corporation for Assigned Names and Numbers (ICANN) that provides access to zone files of a variety of TLDs (including .com, .net and .org). With daily updates to the zone files, it offers the opportunity to study the Domain Name System (DNS) as a whole and to do DNS measurements on a large scale.

It is important for researchers doing such studies (e.g., [2]–[4]) to assess the validity of their measurements, given the age of their data. The correlation between validity and age can vary based on the corresponding TLD: if the TLD has very high churn, more recent data is to be preferred. Futhermore, the development of a TLD as a whole can introduce bias into the patterns that researchers study in their DNS measurement data.

Therefore, it is important to consider the TLD-specific changes within DNS data. In order to examine these differences, we study the development of the old .org TLD (which has been available for almost 40 years [5]) and compare it to the development of a much more recent TLD, .dev (which has only been made available in 2019 [6]). We analyze how both of these TLDs changed from 2019 until 2023 by investigating the zone files for each day provided by the CZDS.

The key research questions we seek to answer are:

- How did the number of resolvable domains in the .org and .dev TLDs develop over this time span?
- How stable are the TLDs, i.e., how many domains stay present in the zone file vs. how much churn is there?
- What is the typical lifetime of a .org domain vs. a .dev domain?
- If domains are not resolvable during the full considered timespan, how many of them are re-added and how long are the periods in between?

2. Background

The DNS is used for resolving readable domain names to IP addresses on the Internet [5]. It uses TLDs such as .org, .com or .dev to partition the name space of domain names [7]. A name server stores different kinds of records for the set of domain names it is responsible for and replies to DNS queries with this information [7].

TLD name servers maintain information on which second level domains (SLDs) (e.g., tum.de) are registered within the TLD and how to resolve them. The collection of all such records stored on a name server is called a zone file.

Following the introduction of many new TLDs and the associated security risks, easier access to such zone files has been required. Thus, the ICANN introduced the CZDS, which is a service that provides easy access to the zone file of all participating TLDs on a daily basis [8].

Domains are usually registered in the corresponding TLD's registry under contract with ICANN [9]. On behalf of the buyer, the registry then communicates the name servers of the domain to the name server maintaining the zone file of the TLD. This allows DNS clients to find the name servers responsible for the domain, which can then provide information to correctly resolve the domain to the corresponding IP address. In this paper, when we say that a domain is *registered* or *resolvable*, we mean that the domain's name servers are present in the TLD's zone file.

3. Related Work

Past surveys have analyzed the kinds of TLDs participating in the CZDS as well as the results of access requests to the zone files [1], [8].

 doi: 10.2313/NET-2023-11-1_17

While most domains are registered for at least one year, Foremski et. al. [10] analyze domains with shorter lifetimes, which they claim are used for abusive purposes. They investigate the time it takes from the registration of new domains until they "die" as well as the most common "causes of death". They do separate analyses of domain mortality per TLD and find that old TLDs such as `.org` or `.com` experience much less "domain deaths" than the newer TLDs like `.party` or `.work` [10].

Similarly, Affinito et. al. [9] analyze the registration periods of domains for different TLDs over a time frame of ten years. They find that in ten prominent TLDs, 95 % of registration periods are exactly the minimum period of one year. They further identify that the share of malicious domain registrations with shorter periods correlates to the corresponding kind of TLD. The data they use comes from zone files in the Domain Zone Database (DZDB) by CAIDA, which provides a historical record of TLD zone files [9].

This paper focuses instead on only two TLDs in detail and does an in-depth analysis and comparison of the stability and churn of domain names in those TLDs.

4. Methodology

This section describes the zone files and the aggregations we perform with them.

4.1. CZDS Zone Files

The zone file that CZDS provides on a daily basis is a compressed, tab-separated file with one entry per line, sorted by domain. Each entry is comprised of the domain name, the cache TTL, the class (usually "IN" for Internet), the type of record (especially relevant for this paper "NS" records for the name servers of a domain) as well as the value of the record (for "NS", this is the domain name of the name server responsible for the domain).

An uncompressed `.org` zone file for a single day in May 2023 is about 1.8 GB large and contains almost 30 M rows. An uncompressed `.dev` zone file for the same day spans 2.1 M entries at 234 MB. For comparison: a zone file of the largest TLD (`.com`) on the same day is 25 GB large and contains over 410 M rows.

Thus, the raw data of the `.org` TLDs for the investigated timespan from November 2019 till May 2023 (1270 days in total) amounts to over 2 TB alone. Efficiently processing and analyzing such large amounts of data in a reasonable amount of time and with reasonable computing resources poses a technical challenge, which is discussed in more detail in the following sections.

4.2. Data Aggregation

To answer the research questions of this paper, computationally intensive aggregations of the raw zone file data are needed. In principle, one could execute all required queries on the raw data itself. Our approach however was to first transform the zone files into a more compact and aggregated form, materialize that form, and then perform our analyses on the aggregates instead. This allowed us to iterate on the downstream analysis tasks more quickly, without constantly perfoming the low-level aggregations.

The two types of aggregates we materialized are: 1) Distinct domains: We de-duplicate the NS records in the zone file by domain and then count the unique number of domains present in the zone file on each day.
2) Registration periods: For each time frame of consecutive days on which a domain was present in the zone file, we store the domain as well as the first and last date of the time frame.

While the first aggregation allows us to investigate the development of the size of the two TLDs as a whole, the second gives us the opportunity to do more fine-grained analyses such as studying the periods between registrations, the registration periods and the churn.

4.3. Data storage

Operating on the raw compressed files would require custom optimized programs to perform the aforementioned aggregations with reasonable computing resources. Thus, we decided to use ClickHouse, an open-source, column-oriented database management system suited for analytical workloads such as our aggregations [11]. It allowed us to simply access the data using SQL.

We imported the raw and compressed zone files provided by CZDS using the `clickhouse-client` CLI to benefit from the built-in parallelization options it offers. With this setup, the import took around 25 seconds per `.org` zone file. Since only NS records are relevant for the research questions of this paper, we filtered the DNS records during the insert process already. By storing the data ordered by domain, we can efficiently partition the full name space and thus make aggregation queries more efficient.

4.4. Challenges

While implementing the pipeline from raw CZDS zone files to final aggregates, we faced several challenges, which are discussed below.

4.4.1. Missing Days. Between November 2019 and May 2023, there have been seven days on which our automated CZDS download process did not successfully acquire the current zone files. This led to incorrect registration periods in the corresponding aggregation (one additional period for each missing day). The query assumed that a missing (domain, date)-entry for such a day meant the domain was not resolvable on that day. To solve this, we instead explicitly determined the missing dates and adjusted the query to not consider registration gaps on those dates.

4.4.2. Registration Periods Aggregation. If one considers only one registration period per domain, the aggregation becomes a simple minimum and maximum of the (domain, date)-entries grouped by domain. However, it can happen that a domain is registered for some period, then it is not present in the zone file for a while, and then comes back later.

Since we need to distinguish separate registration periods, our approach was to group the (domain, date)-entries by domain and collect the sorted dates in a list. We then compare the i-th date with the (i+1)-th date in that list and append a running sum of how often the two dates were

not apart by exactly one day (and the day(s) in between are not part of the missing dates, cf. Section 4.4.1). This running sum provides an ID of the consecutive registration period each (domain, date)-entry corresponds to. Finally, we calculate the minimum and maximum date for each domain and period.

By partitioning the `.org` and `.dev` name spaces based on the domain and by indexing entries by the domain (cf. Section 4.3), we were able to restrict the query to only consider domains in pre-computed disjoint partitions. That way, computing the aggregation query for one out of 1000 `.org` partitions only took 24 seconds on average and was well within the resource constraints.

5. Analysis

Using our two exported aggregations, we examine at the size of the `.org` and `.dev` TLDs as a whole and investigate the domain churn.

5.1. Size Of The TLD

As one can see in Figure 1, the total size of the `.org` TLD grew linearly by about 7 % in the investigated 3.4-year timespan. At the same time, `.dev` more than doubled its size. Notably however, in absolute numbers, `.org` grew by around 700 k domains, while `.dev` added less than 200 k dinstinct domains. In May of 2023, `.dev` still only contained a fraction of the domains of `.org` (368 k `.dev` domains vs. 10.7 M `.org` domains). This shows the large difference both in size and growth of these two TLDs.

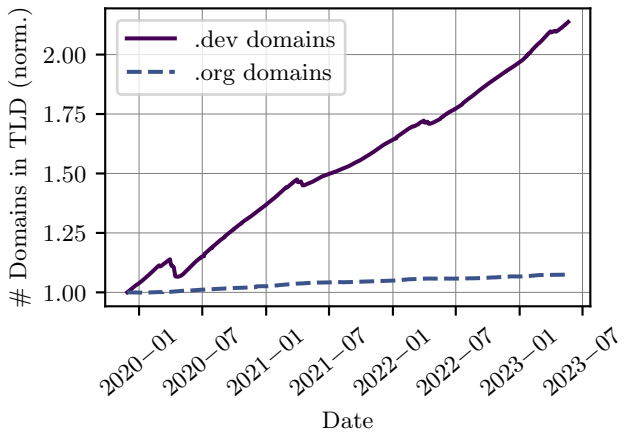


Figure 1: The number of distinct `.org` and `.dev` domains between November 2019 and May 2023. (Both are normalized to the number on November 28, 2019, which is 10 M for `.org` and 172 k for `.dev`)

One further pattern can be observed in the figure: the number of registered `.dev` domains slightly drops in periodic intervals of around one year, each time approximately in April. The magnitude of these drops decreases over the years and is barely visible in 2023. These drops supposedly come from the large amount of initial `.dev` domains registered when the TLD first became available in March 2019 [6]. Domains registrations are mostly sold in one-year increments [9], which is why we observe the small drops exactly one, two, three and four years after the TLD's inception. For `.org`, we do not see this behavior, as it was first available almost 40 years ago [5].

While the number of simultaneously registered `.org` domains increased only by around 7 %, the total number of domains that were present in the `.org` zone file for at least one day during the timespan is around 40 % larger. For `.dev`, the size of the zone file increased by 114 %, but the total number of seen domains is 61 % over that number. This large difference is caused by churn – domains that are registered at some point, and then later again removed from the zone file.

In fact, only 40.4 % of all `.org` domains and 16.1 % of all `.dev` domains ever seen in our data were registered for the full duration. The other 59.6 % of `.org` domains and 83.9 % of `.dev` domains were not present in the zone file on at least one day. We can already see that `.org` displays less churn than `.dev`. Below, we analyze churning domains and the periods in which they were not registered in more detail.

5.2. Periods Between Registrations

Of particular interest are the domains that are registered for some period, are not present in the zone file for a while, and then come back. These domains allow us to study the typical number of days a previously registered domain was not registered for, before it came back. This period represents an attack window for malicious users to publish different name servers and temporarily hijack traffick to a domain [12].

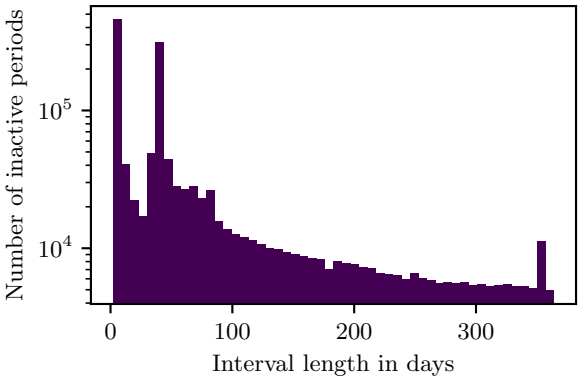


Figure 2: Histogram of the number of days `.org` domains are not present in the zone file after being registered for some duration and before coming back again (logarithmic scale). One vertical bar represents a bucket of 1 week and its height describes the number of periods between registrations we saw whose lengths was this particular number of weeks.

Figure 2 shows a histogram of the duration of these periods in which `.org` domains are not present in the zone file. In total, 1.57 M of such periods from 1.31 M re-appearing `.org` domains were analyzed. Notably, these domains only represent a small fraction of all 10 M `.org` domains. For illustration purposes, the histogram was cut off at a period of 1 year, which represents 88 % of the total data. For `.dev`, we see a very similar pattern, suggesting that the two TLDs do not differ a lot in this regard.

 doi: 10.2313/NET-2023-11-1_17

About a third of periods are less than a week long (most of these – 60 % – are 3 days long). We suspect these do not come from intentional de-registrations of domains, but instead from temporary configuration issues or name server changes.

A second peak occurs at around 35 days. When a domain is not renewed, it is in the redemption grace period status for 30 days, in which it may be re-registered by the previous owner [9]. After that, it takes up to five days until the domain is available to the public again [9]. We assume the second peak comes from domains that were re-registered as soon as they were available again, which is exactly 30 to 35 days after de-registration.

After that, we consistently only see a lower number of period lengths, suggesting that the reason for different inactive durations is just that someone registers the same domain again later. Overall, we can conclude that most attack windows are very short for both `.org` and `.dev`.

5.3. Registration Periods

After analyzing the periods during which domains are not present in a zone file, we study the periods during which domains *are* registered. Figure 3 and Figure 4 show the color-coded fraction of domains still active after a given number of months, differentiated by when the domain was first seen. The first row represents the domains that were present on the first day of our investigated time frame. The other rows represent the domains that were first registered during the month shown on the left (the cohort), while the last row displays an average of all rows. The color in each cell indicates the fraction of domains (between 0 and 1) that are still present in the zone file after the number of months indicated on the x-axis.

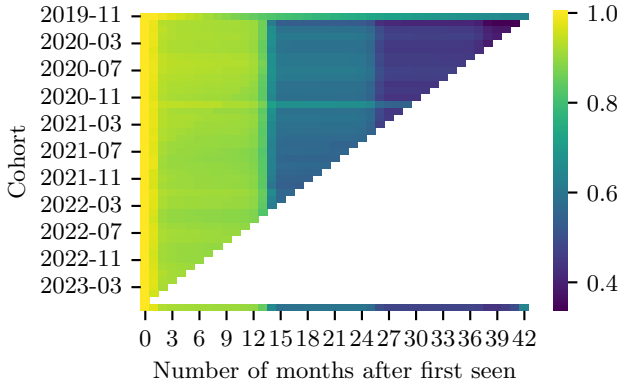


Figure 3: Cohort analysis of `.org` domains from Nov 2019 until May 2023. The last row displays the average over all cohorts.

For both `.dev` and `.org`, one can clearly observe the significant drops in the fraction of registered domains after one, two, and three years for all cohorts except the first – which is unsurprising, given that domain registrations are mostly sold in one year increments [9]. The one- to two-month deviations from the full 12 –month cycle that can be observed in Figure 3 come from auto-renew grace periods during which domains may still be in the zone file.

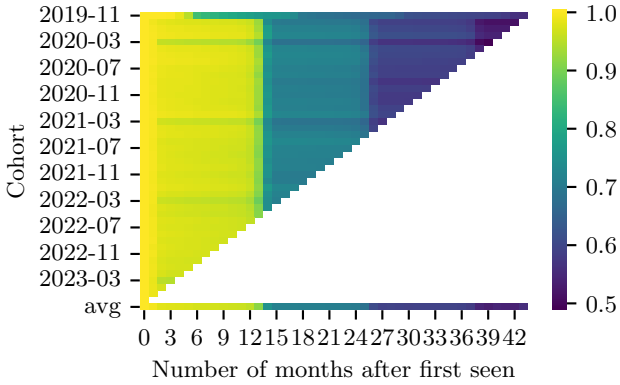


Figure 4: Cohort analysis of `.dev` domains from Nov 2019 until May 2023. The last row displays the average over all cohorts.

The domains represented by the rightmost square in the first row can be considered the "core" of the TLD – these have been in the zone file without interruption since the start of our investigated time frame in 2019. Most notably, 63.6 % of `.org` domains present in the zone file at the start of the investigated time frame are still present at the end. Factoring in that additionally some of the domains that were present at the start but not for the full period might be so because of configuration errors (and the many short 3-day periods between registrations, cf. Section 5.2) we can conclude that the core of `.org` domains is quite stable.

For `.dev`, slightly less domains are there for the full time frame (55.6 %), suggesting that it has a smaller stable core. In the first row of the cohort analysis of `.dev` domains (Figure 4), we can additionally observe the periodic drops in April of each year for the core. This behavior supposedly comes from the considerable share of `.dev` domains that were initially registered when the TLD first became available (as previously discussed in Section 5.1).

6. Conclusion and Future Work

In this paper, we analyzed the development of the old `.org` TLD between November 2019 and May 2023 and compared it to the development of the new `.dev` TLD during the same time span.

We found that the development of `.dev` and `.org` differs in a number of factors. Firstly, `.org` is a much larger TLD, containing 10.7 M domains vs. `.dev`'s 368 k domains in May 2023. Secondly, `.org` shows much less relative growth - it only increased its size by 7 % in the investigated time frame, while `.dev` more than doubled its size. Thirdly, while analyzing the registration periods, we recognized several patterns – the yearly registration period increments as well as `.dev`'s initial registrations in March of 2019. Finally, we showed that `.org` has a larger core of domains that have been registered without interruption since 2019, shows significantly less churn and thus can be considered more stable than `.dev`.

Future work may analyze other large TLDs such as `.com` or a regional TLD such as `.bayern` and investigate differences to `.org` and `.dev`.

 doi: 10.2313/NET-2023-11-1_17

References

[1] A. R. Kang and A. Mohaisen, "Transparency of the new gTLD's centralized zone data service: A measurement study," in *2016 IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 354–355.

[2] J. Zirngibl, S. Deusch, P. Sattler, J. Aulbach, G. Carle, and M. Jonker, "Domain Parking: Largely Present, Rarely Considered!" in *Proceedings of the 6th edition of the network traffic measurement and analysis conference (TMA conference 2022). International Federation for Information Processing (IFIP). https://tma. ifip. org*, 2022.

[3] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld, "Tracking the deployment of TLS 1.3 on the Web: A story of experimentation and centralization," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 3, pp. 3–15, 2020.

[4] J. Zirngibl, P. Buschmann, P. Sattler, B. Jaeger, J. Aulbach, and G. Carle, "It's over 9000: Analyzing Early QUIC Deployments with the Standardization on the Horizon," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 261–275.

[5] J. Postel, "Domain name system structure and delegation," Internet Requests for Comments, RFC Editor, RFC 1591, 3 1994. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1591.html

[6] ICANNWIki, ".dev — icannwiki,," 2019, [Online; accessed 15-June-2023]. [Online]. Available: https://icannwiki.org/index.php?title=.dev&oldid=1170428

[7] P. Mockapetris and K. J. Dunlap, "Development of the Domain Name System," in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM '88. New York, NY, USA: Association for Computing Machinery, 1988, p. 123–133. [Online]. Available: https://doi.org/10.1145/52324.52338

[8] J. Park, J. Choi, D. Nyang, and A. Mohaisen, "Transparency in the New gTLD Era: Evaluating the DNS Centralized Zone Data Service," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1782–1796, 2019.

[9] A. Affinito, R. Sommese, G. Akiwate, S. Savage, K. Claffy, G. M. Voelker, A. Botta, and M. Jonker, "Domain Name Lifetimes: Baseline and Threats," in *6th Network Traffic Measurement and Analysis Conference, TMA 2022*. International Federation for Information Processing (IFIP), 2022.

[10] P. Foremski and P. Vixie, "The modality of mortality in domain names," *Virus*, p. 1, 2018.

[11] I. ClickHouse. ClickHouse - fast open-source OLAP DBMS. [Online]. Available: https://clickhouse.com/

[12] G. Akiwate, R. Sommese, M. Jonker, Z. Durumeric, K. Claffy, G. M. Voelker, and S. Savage, "Retroactive identification of targeted DNS infrastructure hijacking," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 14–32.

Hardware-assisted virtual network benchmarking tools

Eric Rosche, Florian Wiedner*, Christoph Schwarzenberg*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: eric.rosche@tum.de, wiedner@net.in.tum.de, schwarzenberg@net.in.tum.de

***Abstract*—Accurate network emulation plays an essential role in research and development. Most of the time, only a couple of hosts are used to emulate comprehensive networks. This poses difficulties when focusing on singular network characteristics and their tail-behavior such as latency and throughput, as it is difficult to emulate all required aspects realistically. However, this is necessary, especially when looking at e.g. low-latency network emulation in aeronautics, where accurate measurements are crucial. In this paper, we examine how these realistic measurements are achieved by different hardware-assisted network emulators. We achieve this by comparing the different tools and drawing a conclusion from this analysis.**

***Index Terms*—hardware-assisted, network benchmarking, network emulation, low-latency, high-throughput,**

1. Introduction

Network emulation is not a new concept. Common tools like Mininet [1] or NetEm [2] allow the emulation of configurable network topologies and their characteristics. Despite the existance of multiple tools that already allow such emulation, new tools and frameworks continue to be presented. Recent tools like Kollaps [3] and those by Sylla et al. [4], Ryu et al. [5], and Morin et al. [6] for wireless network emulation emphasize the ongoing relevance of this topic. New tools often focus on specific emulation features, such as low latency. When looking at low latency application, fields like Time-sensitive Networking and aeronautics, in which as-fast-as-possible and reliable calculations are crucial, emulation is a valuable tool. Other aspects like complexity and high throughput are interesting to emulate, as the importance of data centers with a huge amount of traffic over short links is only rising with further people and systems transferring to the cloud. These aspects are difficult to emulate only using virtual networking devices and require hardware assistance. This is made possible by specifications like Single-root Input/Output Virtualization (SR-IOV), which allow for Peripheral Component Interconnect Express (PCIe) to be virtualized.

In this paper, we will review multiple of these hardware-assisted tools, focusing on different network characteristics such as low latency and high throughput, and what design choices made this possible. In Section 2 Background about the topic will be provided. Section 3 presents the four different tools analyzed in this paper. Afterwards, we compare the tools in Section 4 and draw conclusions in Section 5.

2. Background and Related work

There are different types of network emulation tools. Some tools, like Mininet, are purely virtual solutions. These kind of tools are designed to emulate a full network on a single host and use features like namespaces for this purpose. Other tools can be defined as testbeds of multiple hosts, which are used on a large scale and by multiple people at once, spanning over one or multiple data centers. These testbeds are in their core real networks designed for emulating real traffic in a controllable environment. Testbeds like FABRIC [7], which was presented 2019, SCIONLab [8], which has been in use since 2016, or PlanetLab [9], established in 2003, are examples in this context.

In this paper, we will focus on a different kind of network benchmarking tool, namely the hardware-assisted variant. The tools analyzed in this paper all share the characteristic that they use real networking hardware to allow benchmarking, and are designed to only use a very limited number of devices. Depending on which hardware is used in the design, this allows for realistic data from real devices. We will also focus on tools which use off-the-shelf hardware.

2.1. Similar comparisons

Different comparisons of network emulation tools have been done before this analysis, though they focus on other aspects. In the article presenting the Kollabs emulator Gouveia et al. [3] present a very recent listing of available virtual network emulation tools. The main focus there lies in the comparisons of features, like dynamic changing of network properties, and in which manner the tools are implemented.

The article presenting the SCIONLab testbed [8] features a short table comparing different network testbeds on their features against the presented SCIONLab. We could not find a recent comparison of these large scale testbeds on a set of universal features. Comparisons like the one made by Mirkovic et. al in [10] from 2011 are older and therefore do not reflect the current iterations of these testbeds.

2.2. Simulators

In comparison to network emulation, network simulation only tries to replicate network behavior. Network simulators like OMNeT++ [11] and NS3 [12] therefore serve a different purpose. Similarly to the comparison

 doi: 10.2313/NET-2023-11-1_18

of network emulators, we were not able to find recent network simulator comparisons. A comparison from 2014 by Kabir et. al [13] shows that with the huge amount of simulators available for different platforms and with different design goals, that a suitable simulator with the required features should be identifiable.

2.3. SR-IOV

SR-IOV is a standard which allows a single PCIe device to be split into multiple virtual devices, which can then be used in virtualization. To achieve this, SR-IOV uses virtual and physical functions. Physical function are complete PICe functions, which allow for configuration of the device in the standard way. Virtual Functions (VF) on the other hand are soly capable of sending and receiving data.

This allows for different Virtual Machines (VM) to use the same PCIe device without any other kind of resource allocation. The hypervisor or the managing operating system of the VMs must support SR-IOV, as the physical function could be required at some point in the virtualization process. This can speed up the virtualization of e.g. networking devices as are used by the tools discussed in this paper [14].

3. Hardware-Assisted Virtual Network Emulation Tools

In the following, we will introduce the tools we will be comparing. All of the following tools provide the ability to benchmark virtual network configurations while utilizing hardware assistance. We have selected these tools for comparison because they employ similar techniques, such as SR-IOV, but they have different approaches and design goals.

3.1. HVNet

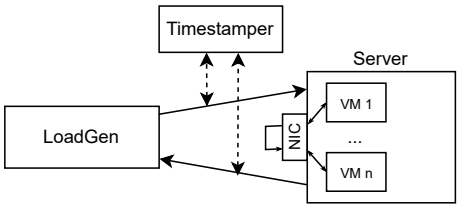


Figure 1: Simplified HVNnet setup

HVNet [15] is an approach for creating virtual network topologies utilizing real networking hardware, with a focus on realistic timestamping of low-latency traffic. To achieve this, a configuration with a load generator that generates traffic, connected to a Device under Test (DuT) using HVNet is used. For the evaluation, timestamping hardware is used to ensure a minimal discrepency. The traffic sent and received by the DuT is channeled over a Network Interface Card (NIC) utilizing SR-IOV to separate it into multiple virtual links. VMs running on the DuT are configured to have minimal overhead. To achieve a setup with VMs instead of using the Data Plane Development Kit (DPDK) [16], like e.g. Mininet is using, a kernel-based networking approach is employed. This setup allows for extremely low-latency traffic emulation, where the 99th percentile of logical link latency is approximately 100 µs for 2-hop measurements at 1 Mbit/s. This tool aims for realistic network behavior, emulating low latency, low jitter network traffic on a easily configurable setup with minimal devices. HVNet setup is summarized in Figure 1.

3.2. NFV-TestPerf

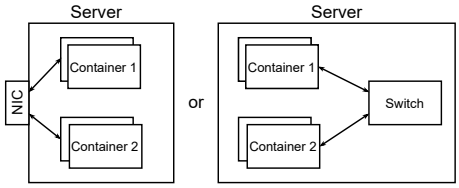


Figure 2: Simplified NFV-TestPerf single host setup

Another way to emulate multiple hosts on a single computer is by using containers. Similar to VMs, containers can be assigned a virtual function on an SR-IOV capable network card. A tool that allows this is NFV-TestPerf [17]. This framework can be used to specify network topologies and configure applications hosted in containers to communicate over different connection types like Linux bridges, virtual switches, or the aforementioned SR-IOV-capable networking hardware. We will focus on the usage of SR-IOV and also discuss the results using VALE [18], a virtual networking switch.

Another feature achieved by this approach, is the framework being very flexible. It allows for single and multiple container communication on single or multiple host configurations, each with different connection types as described above. Six different connection types are compared in multiple scenarios with single or multiple hosts. To achieve network virtualization, DPDK's Application Programming Interface (API) is used. It allows the virtual containers to access the VFs of the SR-IOV capable hardware as well as the virtual software switches. Therefore, the network virtualization runs in user space.

Using this method, tests performed on only a single host with a low packet sending rate and a maximum burst size of 128 achieved latencies below 200 µs using SR-IOV. The NFV-Testperf is summarized in Figure 2.

3.3. TurboNet

A different approach to emulating network behavior with the help of networking hardware is TurboNet [19]. The idea here is to use a programmable switch to emulate multiple configurable virtual switches. The main goal to TurboNet is, in contrast to the two tools mentioned earlier, to emulate switching topologies. To achieve this, the programmable switch is sliced into multiple emulated switches and links. Packets are sent over physical links only when they enter or exit the switch topology.

This is accomplished by assigning each virtual switch a set of ports and emulating link delays via a queue. Additionally, TurboNet allows for multiple programmable switches to be used by connecting them via a physical link.

 doi: 10.2313/NET-2023-11-1_18

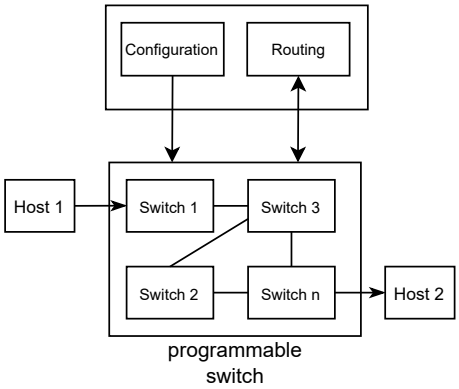


Figure 3: Simplified TurboNet setup

Configuration of TurboNet is available via an API, which also allows for configuring link behaviors such as loss and artificial delays. The TurboNet setup is summarized in Figure 3.

3.4. ExRec

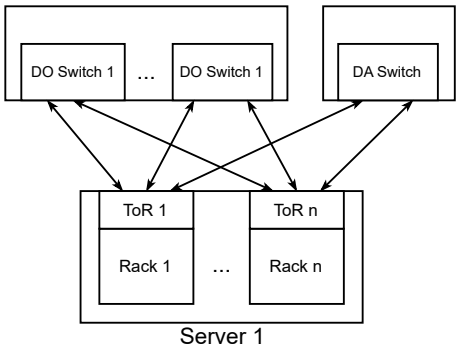


Figure 4: Simplified ExRec single host setup

As data centers become increasingly relevant and continue to grow in size, they represent another crucial application for network emulation. An example of a framework tackling this issue with off-the-shelf hardware is ExRec [20]. In the setup of ExRec, a variable number of hosts M emulates N data center racks, which are connected to top-of-rack switches emulated on the M hosts with virtual machines. The ToRs are then connected to an electric packet switch, which emulates k demand-oblivious spine switches. An optical circuit switch is used as a demand-aware switch. The framework also ensures that network bottlenecks occur only as intended, and that there are no bottlenecks on the VMs. For control messages, the most achievable inter-arrival time was 500 µs, but the focus of the testbed is on high throughput with realistic data center behavior. The ExRec setup is summarized in Figure 4.

4. Comparison of the Tools

In the following, we will try to compare the designs, features, and limitations of the previously presented tools. We will focus on the purpose each tool serves and how they achieve this in comparison to the other tools. The comparison is summarized in Table 1.

4.1. Low Latency

Three of the four mentioned tools have low latency as a requirement or feature. HVNet is specifically designed with low latency in mind, as is NFV-TestPerf. In both cases, SR-IOV capable network cards can be used to transfer packets between different virtual network nodes.

TurboNet uses emulated switches, which only have a nanosecond delay because of the use of loopback ports as links. In the evaluation, a Tofino switch is used for which the dequeuing plus enqueuing delay adds up to around 200 ns for all available bandwidths from 10 Gbit/s to 100 Gbit/s. As this is not a realistic representation of larger networks, TurboNet uses queue depth in programmable switches to bridge this delay to a more realistic value. Even with passing a 10 Gbit/s delayed queue, the delay ranges from 100 µs to 1000 µs.

ExRec does not have a focus on low latency directly. Latency measurements are not directly highlighted in the article describing the tool, but in the evaluation of the testbed, switches are configured at runtime, which is displayed to be possible with an inter-arrival time of about 500 µs. This, however, only gives us a very low bound of what the actual worst case latency of packets at full load could be, which should be significantly higher. For latency, however, only the worst case is an interesting metric, as a big deviation can lead to unreliable results. In this regard, it is, therefore, limited in comparison to the other tools.

When comparing HVNet and NFV-TestPerf, HVNnet with its $\approx$ 200 µs worst-case latency for 2 hops, and NFV-Testperf with a mean latency between 100 µs to 200 µs for a burst rate of 128 and 1500-byte packets, seem to both reach a good performance. It is, however, important to note that virtual networking for HVNet is done in the kernel space, where NFV-Testperf's is done in user space. Additionally, NFV-Testperf uses packets with a size of 1500 bytes, where HVNet uses packets with a size of 363 bytes. But when pulling Mininet as a fully virtual alternative to the test, as has been done in the publication presenting HVNet [15] it has worst-case latencies of about 600x (about 100 ms). This shows that to achieve these kinds of latencies, optimized processes and hardware are absolutely necessary.

4.2. Throughput

Another very interesting network metric is throughput. Especially when emulating multiple virtual hosts on a single real one, high throughput can be difficult to achieve when using real networking hardware. Using an SR-IOV capable network card for multiple virtual links, the maximal achievable throughput for each virtual link is the maximal throughput the network card can handle divided by the amount of virtual links [15]. In more complex emulated topologies, this can quickly become an issue. Looking at a 10 Gbit/s network card, only 10 virtual links can be assigned to physically reach a Gbit/s one-directional link throughput. Additionally, packet size can also have a big influence on throughput, as reducing the packet size by half doubles the cost for the same throughput. The second limit is, therefore, the emulation cost coming from high throughput traffic. This can reduce the emulatable network size in the worst case exponentially

 doi: 10.2313/NET-2023-11-1_18

TABLE 1: Comparison of Tools

Tool	Topology	Networking Hardware	Latency	Throughput
Mininet	fully virtual hosts	-	$\approx$ 200 ms	1 Gbit/s
HVNnet	LoadGen, Timestamper, and DuT	SR-IOV NIC	worst-case $\approx$ 200 µs	N/A
NFV-TestPerf	Host spawning containers	u.a. SR-IOV NIC	mean $\approx$ 100 µs to 200 µs	$\approx$ 13 Gbit/s
TurboNet	Emulated switches in programmable switch	programmable switch	emulated $\approx$ 200 µs to 1000 µs	40 Gbit/s
ExRec	hosts virtualizing racks connected to switches	NIC and emulated switches	min 500 µs	10 Gbit/s

It is important to add that these tools were evaluated in their presenting article and are therefore using different hardware. The displayed values are to be viewed as a reference. Mininet throughput from [21]

if every virtual host would be connected to every other virtual host.

TurboNet tackles this issue by implementing its own background traffic emulation. This is done by using programmable switches to inject packets into the switch pipeline. In a comparison made by Emmerich et al. in the paper presenting MoonGen [22], a popular packet creation tool, it is struggling to reach a comparable kind of throughput for smaller packet sizes.

ExRec is another tool that focuses more on achieving throughput goals, despite the emulation of switches. In their evaluation, the tool could reach high throughput levels extremely fast, right after a certain preconfigured flow had been started.

As NFV-TestPerf's primary goal is to emulate virtualized network functions, which can also need high volumes of data depending on the application, throughput is also a metric interesting here. SR-IOV has some of the best achieved results in the throughput evaluation of the tool, only being beaten by VALE in some scenarios, where throughput reaches up to 12 Gbit/s.

4.3. Results

After comparing the different tools, we can draw some results and requirements for this kind of tools.

4.3.1. Low Latency. For measuring low latency traffic, hardware supporting virtualization appears necessary though measurements using VALE returned good results in the evaluation of NFV-TestPerf [17]. This could be interesting to analyze further for the other tools, especially HVNet as it has a similar structure. In comparison, all the tools achieve latencies that are far below what Mininet and other purely virtual solutions can achieve.

4.3.2. High Throughput. When looking at throughput, a different conclusion can be drawn. As throughput is limited by the hardware, using VALE achieved good results in the evaluation of NFV-TestPerf [17]. The tests where this was conducted were limited to a very small number of containers. It is expected that for more virtualized nodes this result will only strengthen, as then the worst-case exponential limitation of the hardware will come into effect. It would however be interesting to analyze the load that is then put on the CPU, as it would rise in the same manner as the hardware limitation. Overhead from each host also plays a role here, as it doesn't limit the configuration using hardware, but the software switching may be affected. This is why NFV-TestPerf is implemented to only allow network bottlenecks.

4.3.3. Final Points. The comparison with Mininet shows that specialized setups can achieve realistic low latency and also high throughput traffic even when simulating bigger network topologies with only very few hosts. But other than Mininet, which is designed to be as flexible as possible, a lot of additional work is required to allow for more emulation configurations. Tools like ExRec and TurboNet can only emulate a very specific setup and cannot be used to emulate a more general network topology. HVNet and NFV-TestPerf are designed to allow a more general topology but may face challenges when simulating more specific network types, e.g., switching, like TurboNet is designed to, or data center behavior as is done by ExRec. It, therefore, becomes clear that a tool for all purposes, like Mininet, cannot create the best results for specific network aspects, and that this focus requires specialized setups in itself. So if an emulator is to be chosen for a project and the standard tools do not meet the requirements for the emulation, an emulator specialized in the characteristics of the project may be hard to find. Some of the tools that have been compared in this article are available open source, namely ExRec and NFV-Testperf, and the links to GitHub are still active and working at the time of writing.

5. Conclusion and Future Work

In this paper, we compared different hardware-assisted network emulation tools for their features. This allowed us to verify the importance of hardware in emulators where low latency is a design goal. We also learned that focusing on a few characteristics of networks can easily lead to restrictions concerning other characteristics. When using NICs in the emulation process, the number of links that are emulated over it imposes a harsh restriction on the achievable throughput.

However, if no hardware is used, the tools have significant downsides in these regards.

The comparison leaves a few open questions that could be answered in future work. For example, the comparison of the tools was done with a focus on the emulation of low latency networks and throughput. Other, more abstract characteristics of networks could be compared, like the emulation of a network with a high number of nodes or different scenarios like cross-traffic load. Additionally, the tools could be compared in a more practical way by setting them up and running tests on them. Other aspects mentioned in Section 4.3 could also be compared, like the performance of VALE in the other tools.

References

[1] "An Instant Virtual Network on your Laptop (or other PC)," mininet.org, [Online; accessed 14-July-2023].

[2] S. Hemminger, "Network Emulation with NetEm," April 2005.

[3] P. Gouveia, J. a. Neves, C. Segarra, L. Liechti, S. Issa, V. Schiavoni, and M. Matos, "Kollaps: Decentralized and dynamic topology emulation," 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387540

[4] T. Sylla, L. Mendiboure, M. Berbineau, R. Singh, J. Soler, and M. S. Berger, "Emu5gnet: an open-source emulator for 5g software-defined networks," in *2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2022, pp. 474–477.

[5] B. Ryu, R. Knopp, M. Elkadi, D. Kim, and A. Le, "5g-emane: Scalable open-source real-time 5g new radio network emulator with emane," in *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, 2022, pp. 553–558.

[6] D. G. Morin, P. P. ManuelJ. López Morales, and A. G. A. A. Villegas, "FikoRE: 5G and Beyond RAN Emulator for Application Level Experimentation and Prototyping," 2022.

[7] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[8] J. Kwon, J. A. García-Pardo, M. Legner, F. Wirz, M. Frei, D. Hausheer, and A. Perrig, "Scionlab: A next-generation internet testbed," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–12.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[10] J. Mirkovic, A. Hussain, and H. Shi, "A comparative study of network testbed usage characteristics."

[11] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[12] "ns-3 Network Simulator," https://www.nsnam.org, [Online; accessed 14-July-2023].

[13] M. H. Kabir, M. J. H. Syful Islam, and S. Hossain, "Detail comparison of network simulators," vol. 5, no. 20, 2019.

[14] P. Legros, "Why using Single Root I/O Virtualization (SR-IOV) can help improve I/O performance and Reduce Costs," https://www.design-reuse.com/articles/32998/single-root-i-o-virtualization.html, [Online; accessed 02-August-2023].

[15] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "Hvnet: Hardware-assisted virtual networking on a single physical host," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–6.

[16] "Data Plane Development Kit," https://www.dpdk.org/, [Online; accessed 02-August-2023].

[17] G. Ara, L. Lai, T. Cucinotta, L. Abeni, and C. Vitucci, "A framework for comparative evaluation of high-performance virtualized networking mechanisms," in *Cloud Computing and Services Science*, D. Ferguson, C. Pahl, and M. Helfert, Eds. Cham: Springer International Publishing, 2021, pp. 59–83.

[18] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 61–72.

[19] J. Cao, Y. Liu, Y. Zhou, L. He, and M. Xu, "Turbonet: Faithfully emulating networks with programmable switches," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1395–1409, 2022.

[20] J. Zerwas, C. Avin, S. Schmid, and A. Blenk, "Exrec: Experimental framework for reconfigurable networks based on off-the-shelf hardware," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, 2021, pp. 66–72.

[21] A. Al-Sadi, A. Al-Sherbaz, J. Xue, and S. Turner, *Developing an Asynchronous Technique to Evaluate the Performance of SDN HP Aruba Switch and OVS: Proceedings of the 2018 Computing Conference, Volume 2*, 01 2019, pp. 569–580.

[22] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 275–287.

Current State of Hardware and Tooling for SDR

Nico Rumsch, Leander Seidlitz*, Jonas Andre*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: nico.rumsch@tum.de, seidlitz@net.in.tum.de, andre@net.in.tum.de

***Abstract*—Software Defined Radios have become increasingly important because of their unique feature to support a wide multitude of frequencies, modulation modes, amplitudes and waveforms which makes a single device useful for a variety of applications. In the following, the history of SDR and the current state in terms of use cases, applications and hardware will be presented. Furthermore, a comparison of GNU Radio, Matlab and SDR# regarding their support for hardware and applications will be made. Lastly, a small overview of the usage of Field Programmable Gate Arrays to improve the performance of Software Defined Radios is given.**

***Index Terms*—software-defined radio, history, hardware, software, fpga, gnu radio**

1. Introduction

In the 1970s a need for more easily configurable radios started to arise. Before the introduction of Software Defined Radios (SDRs), it was always necessary to implement the ability to, for example, receive different frequencies by using more hardware components. With SDR a hardware and software platform was created to solve this problem. Its possibility to be controlled by software to receive and transmit different radio signals without requiring modifications to the hardware itself makes it a versatile tool for any radio hobbyist or researcher. This holds especially true for cellular network research where hardware can be used to simulate different networks. In an ideal scenario, a single hardware device can be configured by software to transmit or receive any imaginable frequency, or waveform, at any data rate. However, current hardware offers have physical limits and operate in specific boundaries like frequency ranges. [1]

1.1. History

The first step towards the SDRs known today was done by Joe Mitola when he defined the term Software Radio (SR) in 1992 as a system that consists of a Radio Frequency (RF)-frontend, Analog Digital Converter (ADC) and Digital Signal Processor (DSP). His proposed architecture alleviated the hardware responsibility of decoding the signal and moves it to a dynamically configurable DSP which thereby can easily support different waveforms or frequencies amongst others. [2] The term "Software Defined Radio" was later introduced by Stephen Blust in 1995 [3]. Before the term existed, already in 1984 E-Systems Inc. implemented the first SDR in 1984 [4]. Four years later, in 1988, researchers of the Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt (DFVLR), the predecessor of today's Deutsches Zentrum für Luft- und Raumfahrt (DLR), developed the first SDR transceiver which could be configured through software as part of a digital satellite modem [5], [6]. Later in the 1990s, the first large-scale application of a SDR platform was deployed by the US military, called SpeakEASY 1 and the next generation SpeakEASY 2 [7], [8]. After 2000, SDRs matured to a point where no significant change to the concept occurred. All newer developments are in improved performance, smaller chip sizes, lower power consumption and better affordability. For hobbyists, a range of offers for cheap SDR receivers emerged, while researchers benefit from better performance and a wider range of applications.

1.2. Functionality of SDRs

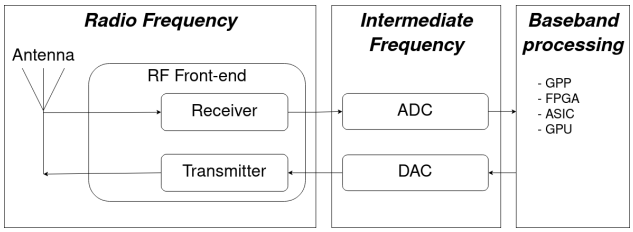


Figure 1: Example SDR architecture [9]

Figure 1 describes a common architecture for a SDR with the three main components being a radio frequency frontend, a converter between digital and analog signals and a processor for interpreting or producing the digital values. The RF-frontend is responsible for receiving the radio wave from the transmission medium. This incoming signal is sampled twice with one sample being phase shifted by 90 degrees which is called I/Q sampling and simplifies the hardware requirements for the SDR. [10]

I/Q sampling works on the premise that any waveform can be reconstructed by adding the amplitude of a sine and a cosine wave [10]. This gives an unambiguous waveform, which is not possible to achieve with the multiplication of in-phase waves [11]. Furthermore, representing a wave with the amplitude I and Q is easier than working with the amplitude and phase of a wave. The final formula to represent a wave can be seen in Equation 1. Here, frequency is abbreviated by f, total wave amplitude by

 doi: 10.2313/NET-2023-11-1_19

A, time by z, phase by φ, In-phase amplitude by I, and Quadrature amplitude by Q. [12]

$$A \cdot \cos(2\pi f t + \varphi) = (\sqrt{I^2 + Q^2}) \cos(2\pi f t + \arctan(\frac{Q}{I})) \tag{1}$$

In the formula, the cos component is called the carrier because it is the base wave onto which the information will be encoded. Changing the I and Q amplitude of the sine and cosine wave is called modulating and encodes said information. [10]

The two sampled amplitudes are converted by the ADC to discrete digital values or created from digital values in case of transmission by the Digital Analog Converter (DAC). When sampling a frequency it is necessary to sample at double the wanted frequency, called the Nyquist frequency, to fully capture the characteristics of it. For example, in the case of Bluetooth's and WiFi's $2.4\,\mathrm{GHz}$ signals, it is required to sample at 4.8 GSps which in turn can only be achieved with expensive ADCs/DACs. To solve this problem, the incoming signal is converted into an intermittent frequency before it is digitalized. This allows for the removal of the carrier from the frequency, which creates a signal for the ADC/DAC that is centered around $0\,\mathrm{Hz}$. This is known as the baseband. [10]

The final I/Q values from the baseband signal can then be used by the processor, which may be for example a General Purpose Processor (GPP), a Field Programmable Gate Array (FPGA), a Graphics Processing Unit (GPU), or an Application-Specific Integrated Circuit (ASIC) to decode the signal using available software.

2. State of the art

With advances in the hardware and software field, modern SDR systems can, amongst others, operate on a wider range of frequencies, and support more modulation types and waveforms. In the last two decades, many new systems were created that benefit from this dynamic usage of hardware. One of the most prominent examples is the development of cellular network standards (3G, 4G, 5G) [3]. Especially for 4G and 5G, the possibility for a software-defined base station enables faster development and updates to the mobile network with only software changes [13].

2.1. SDR use cases

This section goes more into detail about cellular networks and describes the usage of SDRs in amateur radio.

2.1.1. Cellular networks and wireless communication. Since the 4th generation of cellular networks, Software Defined Networks are playing a more important role by allowing for easy reconfigurability and dynamic deployments. This however does not yet extend to the physical layer of the cell towers, where current research is proposing to integrate SDRs as the missing building block. [13]–[15]

In contrast, researchers and developers already adopted SDRs for the field of cellular networks or wireless communication which can be seen in Table 5, where two of the three presented tools support the simulation of cellular networks based on SDRs.

2.1.2. Amateur radio. SDRs play an important role in amateur radio. Anyone can receive a wide range of frequencies and modulations, which makes SDRs popular among hobbyists. This interest is further increased by the option for affordable hardware, like the RTL-SDR family.

One popular area, where many enthusiasts use SDRs, is in the context of aircraft positional data (Automatic Dependent Surveillance - Broadcast (ADS-B)). It is a public, worldwide system where almost every commercial aircraft broadcasts unencrypted details about itself, amongst other information its position, on the $1090\,\mathrm{MHz}$ frequency. Many community-based projects use local, terrestrial SDRs to receive the data and send them to a message broker, which combines all the received data and provides a (public) dataset or live Application Programming Interface (API). [16]–[19]

Other example use cases are RF fingerprinting, spectrum analysis, drone detection or decoding in general [20]. For a selection of further protocols which can be freely received see the subsequent Section 2.2.

2.2. Protocols

The benefit of SDRs is the support for many different applications and protocols via one single device. They can be used by all computers or laptops, as the devices are, in most cases, accessible via the Universal Serial Bus (USB) or network. A selection of applications and their corresponding frequency ranges can be found in Table 1.

2.3. SDR hardware

In Table 3 ten SDRs are presented with their hardware specification. Compared are the available processor types for which a more detailed comparison can be found in Table 2. A SDR with an onboard FPGAs has the potential to move some of the program logic onto the same for better performance, as described in Chapter 3.2. This makes an onboard FPGAs a key feature to consider. Furthermore, it is indicated if the hardware is a receiver, transmitter or transceiver and in which configuration it can be operated, primarily half-duplex, full-duplex or both. For the ADC and DAC the sampling rate and bit depth are listed where it is better to have higher values in each category. The bit depth indicates how precise a signal can be received or reconstructed and the sampling rate of how often this conversion can happen per second. The same applies to the overall sampling rate, which might differ between the ADCs and DAC. This can be caused by slower performance in, for example, the communication interface onboard processor. The frequency spectrum indicates which frequency signals can be received or transmitted. A wide frequency range, which goes into the upper and lower bound extremes, is better than an SDRs with a more narrow range. The bandwidth, as can be seen in figure 2, specifies which surrounding frequency range can be observed around the tuned-to frequency of the SDR.

 doi: 10.2313/NET-2023-11-1_19

TABLE 1: Protocols or applications in frequency ranges [21], [22]

Frequency range	Application	Protocol
30-300 kHz	Navigation	
300 kHz-3 MHz	Marine/Aircraft navigation, AM broadcast	
3-30 MHz	Broadcasting, mobile radio	NFC/EMV, Automatic Identification System (AIS)
30-300 MHz	FM radio broadcast	
300 MHz-1 GHz	Cell phones, mobile radio, Internet of Things (IoT), TV	LoRa, SIGFOX, ZIGBEE, Z-Wave, DVB-T2
1-3 GHz	WLAN, Cell phones, IoT	ZIGBEE, ANT+, WIFI, LoRa, Bluetooth, ADS-B
3-60 GHz	Radar, Cell phones	WIFI(6)

TABLE 2: Technologies for signal processing on SDR [21], [23]

Type	Performance	Power	Size
GPP	low/medium	low	medium
DSP	medium	medium	large
FPGA/SoC	high	medium	large
ASIC	high	low	small

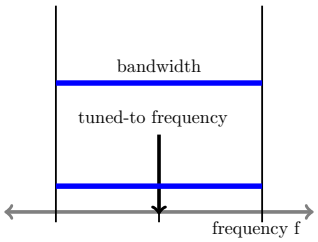


Figure 2: Explanation: Bandwidth

2.3.1. Receiver. In the hobbyist space receivers are popular because of their affordability with a multitude of platforms, such as RTLSDR and Airspy, being available. The typical frequency range of these systems is between 1 MHz and 2 GHz with varying resolutions and amounts of DACs/ADCs. Devices supporting this range can therefore already receive more than half of the applications and protocols mentioned in Table 1. The number of samples per second increased in the past to about 200 MSps for expensive systems, with some more exotic products being able to achieve rates in the range of GS/s by using a high-performance FPGA for the processing, combined with high-performance ADCs and DACs. [21]

2.3.2. Transceiver. Transceivers are devices that can both send and receive data. Generally speaking, the transmitter part of an SDR transceiver is either equally or less powerful than the receiving part. Compared to a device, which can only receive, the receivers on a transmitter are more powerful than their receive-only counterparts. This includes, amongst other, support for larger frequency ranges, higher bandwidths, higher samples per second and better resolutions of the ADCs/DACs. [21] Following the same trend, professional-grade transceivers have the same benefit over consumer hardware with microcontrollers, custom System on a Chip (SoC), FPGAs or even GPUs (AIR-T [36]) for very high throughputs. Popular consumer transceivers are from HackRF and LimeSDR, while in the professional space devices from USRP are popular.

3. Software support for SDR protocols and hardware

Over the years, hardware and software for different use cases of SDRs got developed. Software for SDRs connects to the hardware at the baseband processing step (see Figure 1). The following section will present a selection of software, their support for the previously presented hardware, and support for different use cases.

3.1. Software

The available software for SDRs can range from simple command line tools to graphical user interfaces, sometimes with support for protocol-specific visualizations. There are over 30 universal and even more single-purpose tools available for the popular RTL-SDR platform alone. [37]

3.1.1. GNU Radio. One of the most popular tools is GNU Radio originally released by Eric Blossom in 2001 as an official GNU project. It is being continuously developed and uses the concept of flowgraphs to define block-based transformations. While all processing operations are implemented in C++, the definition of the flowgraphs can be written in either C++ or Python. GNU Radio is supported on Linux, Windows, and MacOS. [38]–[40]

3.1.2. Matlab and Simulink. Matlab and Simulink are proprietary software products by MathWorks. The suite supports a wide range of applications from linear algebra and numeric computing to complex simulations. [41] Amongst others, it offers functionality to simulate wireless networks directly on hardware. For this, the software can interface with a wide range of platforms and communicates with the digital processing unit of the SDR. Matlab is supported on Linux, Windows, and MacOS. [42]

3.1.3. SDR#. SDR# is a simple-to-use, general-purpose visualization tool for SDRs, running on Windows only. It visualizes real-time readings of the frequency and spectrum from the SDR. [43] Furthermore, it has a rich plugin system to enable support for more protocols and SDR applications. [44] Hardware-wise it natively only supports the Airspy platform but is extended by official or community-developed plugins and can interface with a wide variety of SDRs. [45]

3.2. Hardware

The support of GNU Radio, Matlab and SDR# for hardware devices, as presented in Table 3, is indicated in

 doi: 10.2313/NET-2023-11-1_19

TABLE 3: Comparison of selected SDR hardware

Hardware	Chipset	Processor Type	RF Frontend	Receiver/Transmitter	Duplex	ADC/DAC resolution [Bits]	ADC/DAC sampling rate [MSps]	Sampling rate [MSps]	Frequency range [MHz]	Bandwidth [MHz]	Interface
RTL-SDR [24]	R820T2	n/a	n/a	1/-	n/a	8/-	n/a	28.8	0.5 – 1766	2.4	USB
Airspy R2 [25]	R860	GPP	35dBm IIP3	1/-	n/a	12/-	20/-	10	24 – 1700	9	USB
Airspy Mini [26]	R860	n/a	35dBm IIP3	1/-	n/a	12/-	36/-	10	24 – 1700	6	USB
LimeSDR [27]–[29]	LMS7002M	FPGA	n/a	2/2	full	12/12	n/a	61.44	0.1 – 3800	61.44	USB
LimeSDR PCIe [28]–[30]	LMS7002M	FPGA	n/a	2/2	full	12/12	n/a	61.44	0.1 – 3800	61.44	PCIe
HackRF One [21], [31]	MAX2837	GPP,FPGA	n/a	1/1	half	8/10	20/20	20	1 – 6000	20	USB
USRP B200 [32]	AD9364	FPGA	20dBm IIP3	1/1	both	12/12	61.44/61.44	61.44	70 – 6000	56	USB
USRP B210 [33]	AD9361	FPGA	20dBm IIP3	2/2	both	12/12	61.44/61.44	61.44	70 – 6000	56	USB
USRP N320 [34]	n/a	GPP,FPGA	17dBm IIP3	2/2	both	14/16	250/250	250	3 – 6000	200	Ethernet
Per Vices Cyan [35]	n/a	GPP,FPGA	n/a	1-16/1-16	both	16/16	1000/1000	1000	<18000	1000	Ethernet

Table 4. A unique feature of the USRP devices is, that all of them can be controlled with the common interface library USRP Hardware Driver (UHD) [46]. This means, developers can support all devices from this vendor by implementing support for the UHD interface.

While GNU Radio supports all in Table 3 listed devices, both Matlab and SDR# only support a subset of them. Generally speaking, Matlab is focussing more on professional-grade products, in this case from USRP/Ettus Research, and SDR# supports hardware targeted towards enthusiasts.

TABLE 4: Software support for presented hardware [21], [47], [48]

	GNU Radio	Matlab	SDR#
RTL-SDR	Yes	Yes	Yes
Airspy R2	Yes	No	Yes
Airspy Mini	Yes	No	Yes
LimeSDR	Yes	No	Yes
LimeSDR PCIe	Yes	No	Yes
HackRF One	Yes	No	Yes
USRP B200	Yes	Yes	No
USRP B210	Yes	Yes	No
USRP N320	Yes	Yes	No
Per Vices Cyan	Yes	Yes	No

A recent development to achieve even higher performance in SDRs is to utilize FPGAs and implement processing logic in hardware. Three approaches are possible to utilize FPGAs for processing:

1) Build a SDR out of a RF-frontend and FPGA
2) Additional FPGA as an accelerator
3) Utilize SDRs built-in FPGAs

Following the first approach, an implementation of the IEEE 802.11 standard and ZigBee is described in [49]. In contrast, the authors of [50] propose a framework to utilize a FPGA as an accelerator in combination with GNU Radio and show promising results. Lastly, in [8] the authors describe the communication with the host as one of the major limitations of high performance SDRs, which follows the third approach. The authors were able to achieve an up to 64 times higher data rate by moving the demodulation of the signal onto the FPGA and thereby eliminating most communication overhead. The interaction with the FPGA could be performed by a utility from UHD.

Besides higher data rates, another benefit of using FPGAs is faster development times compared to ASICs. However FPGAs also require larger board sizes as more additional hardware is needed. [21], [51]

3.3. Protocol support

As most transmitted data is encoded, software is typically required to decode them and provide specific visualizations, e.g. showing data on a map in the case of ADS-B or AIS.

The support for the in Section 3.1 presented software and the previously presented applications are described in Table 5. Because SDR# itself mainly supports the visualization of the baseband data, it relies on extensions to support applications beyond the build functionality and does not support most wireless communication protocols. Matlab on the other hand has tools for most communication protocols. Lastly, GNU Radio supports a wide range of applications because of its flow graph design and big community support.

In Table 5, support for a protocol will be annotated with "Yes", if additional software is required a "*" will be added and no information available will be indicated by "n/a".

TABLE 5: Protocol support of SDR software [52]–[67]

	GNU Radio	Matlab	SDR#
WIFI	Yes	Yes	n/a
ADS-B	Yes	Yes	Yes*
FM radio broadcast	Yes	Yes	Yes
LoRa	Yes	n/a	n/a
ZigBee	Yes	n/a	n/a
AIS	Yes	Yes	Yes*
3G	Yes	n/a	n/a
4G	Yes	Yes	n/a
5G	n/a	Yes	n/a
Bluetooth	Yes	Yes	n/a

For cellular networks, there also exists the Open Air Interface (OAI) developed by the OpenAirInterface Software Alliance. Its goal is to lower the adoption barrier for Radio Access Networks (RANs) by offering implementations for modern cellular network types like 4G and 5G. [68]

4. Conclusion

Software Defined Radio supports a wide variety of different use cases. It is especially dominant in cellular networks in its recent versions, to enable dynamic upgrades without requiring hardware changes. For enthusiasts, it allows an easy start in the field of radio networks, because a single device can be used versatilely in terms of frequency and application support, while still being affordable. Because of the same features SDRs are largely adopted in development and research to, for example, simulate cellular networks, WIFI or other protocols without requiring specific hardware.

The field is under continuous development to achieve the ideal SDR with its latest development being the inclusion of FPGAs in the SDR. While they have been used as an accelerator for quite a while now, using the SDR's

onboard FPGA is rather new and can greatly improve performance by reducing the communication volume between SDRs and hosts.

References

[1] T. Ulversoy, "Software Defined Radio: Challenges and Opportunities," vol. 12, no. 4, pp. 531–550.

[2] J. Mitola, "Software Radios-Survey, Critical Evaluation and Future Directions," in *[Proceedings] NTC-92: National Telesystems Conference*, pp. 13/15–13/23.

[3] R. Sahu, "Theoretical and Practical Approach to GNU Radio and LimeSDR Platform."

[4] M. T. Mushtaq, M. S. Khan, M. R. Naqvi, R. Khan, M. A. Khan, and O. Koudelka, "Cognitive Radios and Cognitive Networks: A short Introduction," 2013.

[5] R. G. Machado and A. M. Wyglinski, "Software-Defined Radio: Bridging the Analog–Digital Divide," vol. 103, no. 3, pp. 409–423.

[6] Peter Hoeher and Helmuth Lan, "Coded-8PSK Modem for Fixed and Mobile Satellite Services Based on DS," in *Coded-8psk Modem for Fixed and Mobile Satellite Services Based on DSP*, vol. January 1990, pp. 117–123.

[7] R. Lackey and D. Upmal, "Speakeasy: The Military Software Radio," vol. 33, no. 5, pp. 56–61.

[8] S. S. Hanna, A. A. El-Sherif, and M. Y. ElNainay, "Maximizing USRP N210 SDR Transfer Rate by Offloading Modulation to the On-Board FPGA," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pp. 110–115.

[9] T. Juhana and S. Girianto, "An SDR-based Multistation FM Broadcasting Monitoring System," in *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*. IEEE, pp. 1–4, accessed 2023-05-29. [Online]. Available: http://ieeexplore.ieee.org/document/8272943/

[10] Dr. Marc Lichtman, "3. IQ Sampling — PySDR: A Guide to SDR and DSP using Python," accessed 2023-06-10. [Online]. Available: https://pysdr.org/content/sampling.html

[11] Mikael Q Kuisma, "I/Q Data for Dummies," 03/10/2023, 8:08:56 PM, accessed 2023-06-11. [Online]. Available: http://whiteboard.ping.se/SDR/IQ

[12] Dr. Marc Lichtman, "3. IQ Sampling — PySDR: A Guide to SDR and DSP using Python — Carrier Down Conversion," accessed 2023-06-10. [Online]. Available: https://pysdr.org/content/sampling.html#carrier-and-downconversion

[13] H.-H. Cho, C.-F. Lai, T. K. Shih, and H.-C. Chao, "Integration of SDR and SDN for 5G," vol. 2, pp. 1196–1204.

[14] D. Kafetzis, S. Vassilaras, G. Vardoulias, and I. Koutsopoulos, "Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions," vol. 10, pp. 9989–10 014.

[15] F. Xu, H. Yao, C. Zhao, and C. Qiu, "Towards next Generation Software-Defined Radio Access Network–Architecture, Deployment, and Use Case," vol. 2016, no. 1, p. 264, accessed 2023-05-29. [Online]. Available: https://doi.org/10.1186/s13638-016-0762-6

[16] Flightradar24, "Live Flight Tracker - Real-Time Flight Tracker Map," Flightradar24, accessed 2023-05-31. [Online]. Available: https://www.flightradar24.com/

[17] "The OpenSky Network - Free ADS-B and Mode S Data for Research," accessed 2023-05-31. [Online]. Available: https://opensky-network.org/

[18] "Home - Serving the Flight Tracking Enthusiast," ADS-B Exchange, accessed 2023-05-31. [Online]. Available: https://www.dev.adsbexchange.com/

[19] "ADSBHub - Free ADS-B Data Exchange and Plane Tracking," accessed 2023-05-31. [Online]. Available: https://www.adsbhub.org/

[20] W. Jeong, J. Jung, Y. Wang, S. Wang, S. Yang, Q. Yan, Y. Yi, and S. M. Kim, "SDR Receiver Using Commodity Wifi via Physical-Layer Signal Reconstruction," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20. Association for Computing Machinery, pp. 1–14, accessed 2023-06-04. [Online]. Available: https://dl.acm.org/doi/10.1145/3372224.3419189

[21] J. D. J. Rugeles Uribe, E. P. Guillen, and L. S. Cardoso, "A Technical Review of Wireless Security for the Internet of Things: Software Defined Radio Perspective," vol. 34, no. 7, pp. 4122–4134, accessed 2023-06-04. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1319157821000896

[22] L. Hui Fang, S. Hassan, M. AbdulMalek, M. Mazalan, S. Johari, N. Safari, and Y. Wahab, "Development of Microstrip Chebyshev Low Pass Filters using Laser Micromachining."

[23] F. Karray, M. W. Jmal, A. Garcia-Ortiz, M. Abid, and A. M. Obeid, "A Comprehensive Survey on Wireless Sensor Node Hardware Platforms," vol. 144, pp. 89–110, accessed 2023-06-05. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618302202

[24] "Buy RTL-SDR Dongles (RTL2832U)," rtl-sdr.com, accessed 2023-06-18. [Online]. Available: https://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/

[25] "Airspy R2 - airspy.com," accessed 2023-06-14. [Online]. Available: https://airspy.com/airspy-r2/

[26] "Airspy Mini - airspy.com," accessed 2023-06-14. [Online]. Available: https://airspy.com/airspy-mini/

[27] "LimeSDR," Lime Microsystems, accessed 2023-06-14. [Online]. Available: https://limemicro.com/products/boards/limesdr/

[28] "LimeSDR Comparison," Crowd Supply, accessed 2023-06-14. [Online]. Available: https://www.crowdsupply.com/lime-micro/limesdr

[29] "LMS7002M Documentation," MyriadRF, accessed 2023-06-14. [Online]. Available: https://github.com/myriadrf/LMS7002M-docs

[30] "LimeSDR PCIe," Lime Microsystems, accessed 2023-06-14. [Online]. Available: https://limemicro.com/products/boards/limesdr-pcie/

[31] "HackRF One - Great Scott Gadgets," accessed 2023-06-14. [Online]. Available: https://greatscottgadgets.com/hackrf/one/

[32] E. R. Brand, a National Instruments, "USRP B200 USB Software Defined Radio (SDR)," Ettus Research, accessed 2023-06-18. [Online]. Available: https://www.ettus.com/all-products/ub200-kit/

[33] ——, "USRP B210 USB Software Defined Radio (SDR)," Ettus Research, accessed 2023-06-18. [Online]. Available: https://www.ettus.com/all-products/ub210-kit/

[34] ——, "USRP N320," Ettus Research, accessed 2023-06-18. [Online]. Available: https://www.ettus.com/all-products/usrp-n320/

[35] "Cyan – Per Vices," accessed 2023-06-18. [Online]. Available: https://www.pervices.com/cyan/

[36] "Artificial Intelligence Radio Transceiver (AIR-T)," Deepwave Digital, accessed 2023-06-05. [Online]. Available: https://deepwavedigital.com/hardware-products/sdr/

[37] "The BIG List of RTL-SDR Supported Software," rtl-sdr.com, accessed 2023-06-14. [Online]. Available: https://www.rtl-sdr.com/big-list-rtl-sdr-supported-software/

[38] "GNU Radio - The Free & Open Source Radio Ecosystem · GNU Radio," GNU Radio, accessed 2023-06-14. [Online]. Available: https://www.gnuradio.org/

[39] "GNU Radio," accessed 2023-06-14. [Online]. Available: https://en.wikipedia.org/w/index.php?title=GNU_Radio&oldid=1159673658

[40] A. Marwanto, M. A. Sarijari, N. Fisal, S. K. S. Yusof, and R. A. Rashid, "Experimental Study of OFDM Implementation Utilizing GNU Radio and USRP - SDR," in *2009 IEEE 9th Malaysia International Conference on Communications (MICC)*, pp. 132–135.

[41] "MATLAB," accessed 2023-06-14. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MATLAB&oldid=1157953731

[42] "What Is Software-Defined Radio (SDR)?" accessed 2023-06-14. [Online]. Available: https://www.mathworks.com/discovery/sdr.html

[43] J. R. Machado-Fernández, "Software Defined Radio: Basic Principles and Applications," vol. 24, no. 38, pp. 79–96, accessed 2023-06-14. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_abstract&pid=S0121-11292015000100007&lng=en&nrm=iso&tlng=en

[44] "List of SDRSharp Plugins," rtl-sdr.com, accessed 2023-06-14. [Online]. Available: https://www.rtl-sdr.com/sdrsharp-plugins/

[45] "SDR# and Airspy Downloads - airspy.com," accessed 2023-06-14. [Online]. Available: https://airspy.com/download/

[46] "USRP Hardware Driver (UHD™) Software," Ettus Research, accessed 2023-06-14. [Online]. Available: https://github.com/EttusResearch/uhd

[47] "USRP Support from Communications Toolbox," accessed 2023-06-18. [Online]. Available: https://www.mathworks.com/hardware-support/usrp.html

[48] "Airspy@groups.io | USRP / UHD support," accessed 2023-06-18. [Online]. Available: https://groups.io/g/airspy/topic/7621279

[49] A. Di Stefano, G. Fiscelli, and C. Giaconia, "An FPGA-Based Software Defined Radio Platform for the 2.4GHz ISM Band," in *2006 Ph.D. Research in Microelectronics and Electronics*, pp. 73–76.

[50] C. R. Irick, "Enhancing GNU Radio for Hardware Accelerated Radio Design," accessed 2023-06-12. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/33474

[51] M. Petri and M. Ehrig, "A SoC-based SDR Platform for Ultra-High Data Rate Broadband Communication, Radar and Localization Systems," in *2019 Wireless Days (WD)*, Apr. 2019, pp. 1–4.

[52] T. Vilches and D. Dujovne, "GNUradio and 802.11: Performance Evaluation and Limitations," vol. 28, no. 5, pp. 27–31.

[53] cloud9477, "Gr-ieee80211," accessed 2023-06-16. [Online]. Available: https://github.com/cloud9477/gr-ieee80211

[54] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, "Decoding IEEE 802.11a/g/p OFDM in Software Using GNU radio," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. Association for Computing Machinery, pp. 159–162, accessed 2023-06-16. [Online]. Available: https://doi.org/10.1145/2500423.2505300

[55] M. Hostetter, "Gr-adsb," accessed 2023-06-16. [Online]. Available: https://github.com/mhostetter/gr-adsb

[56] S. Meshram and N. Kolhare, "The Advent Software Defined Radio: FM Receiver with RTL SDR and GNU radio," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 230–235.

[57] D. Valerio, "Open Source Software-Defined Radio: A Survey on GNUradio and its Applications," accessed 2023-06-16. [Online]. Available: https://www.semanticscholar.org/paper/Open-Source-Software-Defined-Radio%3A-A-survey-on-and-Valerio/90cdfd630dabf4ea75aea53bbc9c22ae2367e737

[58] B. Oumimoun, L. Nahiri, H. Idmouida, A. Addaim, Z. Guennoun, and K. Minaoui, "Software Defined AIS Receiver Implementation Based on RTL-SDR and GNU Radio," in *2022 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pp. 1–5.

[59] "GR-Bluetooth," Great Scott Gadgets, accessed 2023-06-16. [Online]. Available: https://github.com/greatscottgadgets/gr-bluetooth

[60] "WLAN Toolbox," accessed 2023-06-18. [Online]. Available: https://www.mathworks.com/products/wlan.html

[61] W. Alqwider, A. Dahal, and V. Marojevic, "Software Radio with MATLAB Toolbox for 5G NR Waveform Generation," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 430–433.

[62] "ADS-B and AIS - MATLAB & Simulink," accessed 2023-06-18. [Online]. Available: https://www.mathworks.com/help/comm/ads-b-and-ais.html

[63] "FM Broadcast Receiver - MATLAB & Simulink Example," accessed 2023-06-18. [Online]. Available: https://www.mathworks.com/help/supportpkg/rtlsdrradio/ug/fm-broadcast-receiver.html

[64] "Bluetooth LE Waveform Reception Using SDR - MATLAB & Simulink," accessed 2023-06-18. [Online]. Available: https://www.mathworks.com/help/bluetooth/ug/bluetooth-low-energy-receiver.html

[65] "ADSB# Plugin for SDRSharp," rtl-sdr.com, accessed 2023-06-18. [Online]. Available: https://www.rtl-sdr.com/adsb-plugin-for-sdrsharp/

[66] "Getting Started with RTL-SDR and SDR-Sharp and CubicSDR," Adafruit Learning System, accessed 2023-06-18. [Online]. Available: https://learn.adafruit.com/getting-started-with-rtl-sdr-and-sdr-sharp/sdr-number-fm-radio

[67] J. Demel, S. Koslowski, and F. K. Jondral, "A LTE Receiver Framework Using GNU Radio," *Journal of Signal Processing Systems*, vol. 78, no. 3, pp. 313–320, Mar. 2015.

[68] "OpenAirInterface – 5G Software Alliance for Democratising Wireless Innovation," accessed 2023-06-16. [Online]. Available: https://openairinterface.org/

Temporal Graph Neural Networks

Erik Söhner, Max Helm*, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: e.soehner@tum.de, helm@net.in.tum.de, jaeger@net.in.tum.de

***Abstract*—Graph Neural Networks (GNNs) have become a fundamental tool for working with graph-structured data. They have shown high accuracy in making predictions in a wide range of applications and are now playing an important role in the field of machine learning. To combine the graph structure with temporal data, the Temporal Graph Neural Network emerged, which brings improved predictive performance to a variety of tasks. The paper shows common patterns in temporal and traditional graph neural network architectures and provides information on applications and open challenges.**

***Index Terms*—Temporal Graph Neural Networks**

1. Introduction

Neural networks play an important role in machine learning. These models help machine learning make accurate and efficient predictions [1]. Their ability to recognize patterns allows them to process data where traditional machine learning algorithms struggle due to the complexity of the data. As a result, many types of neural networks are used in a wide range of real-world applications. These include image and speech recognition, natural language processing, autonomous vehicles, and personalized recommendation systems [1]. For graph-structured data, graph neural networks have emerged, contrary to the traditional neural networks that are prevalent for grid-like data. As neural networks show promise for more complex tasks, new neural network architectures are rapidly being developed. Temporal Graph Neural Networks (TGNN) are one of the latest NN architectures. They extend the graph neural network with additional modeling of temporal dependencies on spatial features. In this paper, we give some background on machine learning algorithms, explain graph neural network architectures, and show how temporal graph neural networks extend static GNN architectures. We discuss the possibilities of TGNNs with new frameworks such as PyTorch Geometric Temporal, as well as future challenges in this area of research. The paper also presents a TGNN architecture used for Internet traffic prediction, as well as an overview of other TGNN applications.

2. Background on Machine Learning Algorithms

The most basic type of a Neural Network is the Multi Layer Perceptron (MLP). It only consists of fully-connected layers, namely an input layer, an output layer and hidden layers in between. All nodes of a layer have weighted edges, the learnable parameters, to all nodes of their neighbor layers. The information of a node becomes the weighted average of the node information from the previous layer put in an activation function. This is known as message passing. [2]

Another popular type is Convolutional Neural Network (CNN). They consist of a number of convolutional layers and a fully connected layer to generate the output. As input, it takes a multidimensional array, typically an image with its dimensions: height, width, color channels. The convolution layer computes the convolution operation on the layer's input. This operation first applies filters containing the learnable parameters, a matrix multiplication, then it applies an activation function. Between the layers, the sizes and number of dimension can change, for the fully connected layer, it is arranged in single-dimension array. [3]

2.1. Training of Neural Networks

The training of neural networks is an iterative process in which the output of the neural network is used to make a meaningful change in the learnable parameters of the network. First, a cost function measures the discrepancy between the network's actual output and its optimal output. The optimal output can either be known or the plausibility of the actual output can be checked, depending on the learning method of the neural network. Then, a vector of optimal changes for all learnable parameters is computed to minimize the cost function, called the gradient, and applied to the parameters. [4]

3. Graph Neural Networks

Traditional neural networks perform well on a wide range of tasks, but leave a lot of potential when working with graph-structured data. Therefore, GNNs are designed to exploit the existing dependencies. Consequently, GNNs have shown superior performance in a number of domains where they outperform other machine learning (ML) algorithms. Examples of domains where GNNs are successfully used are social network analysis, drug discovery, or recommendation systems. [5]

The architecture of GNNs can be broadly classified into spatial and spectral GNNs [6]. Architectures of GNNs are highly variant and difficult to generalize. The following descriptions of architectures can be understood as feasible architectures that use common functions and patterns across various GNN architectures.

 doi: 10.2313/NET-2023-11-1_20

3.1. Spatial Graph Neural Networks Architecture

In spatial GNNs, each layer updates the graph representation. Here we assume a traditional GNN with a static graph, so the only thing that changes in each layer are the node features. The message passing of a hidden layer is processed for each node individually. The neighbor node features are the input for a message-passing function, which can be decomposed into four functions that may occur. [6]

1) **Transformation function** is a matrix multiplication on each input's node feature map. This is where attention mechanisms, a multiplication by the corresponding edge's weight, are usually located. In general, this weight matrix can contain learnable or static parameters or edge features.
2) **Aggregation function** takes the same feature from all transformed node feature maps. The aggregation is applied to each feature, creating a single feature map. Typical aggregation functions are min, max, sum, or average.
3) **Update function** combines the remaining features from previous operations with the node features in element-wise operations.
4) **Activation function** amplifies the node features. Typical activation functions include sigmoid, ReLU, and softmax.

3.2. Spectral Graph Neural Networks Architecture

A major difference to spatial GNNs is the global nature of message passing in the spectral domain. In order to operate in the spectral domain, the feature matrix must first be transformed by multiplying it by the eigenvector of the graph's Laplacian matrix. Then, the functions present in a layer can be applied. [6]

1) **Convolution** is the multiplication of the transformed matrix with a diagonal matrix of weights
2) **Activation function**
3) **Pooling** takes a single value from a neighborhood of nodes, reducing the dimension of the feature map. Typical pooling functions are max pooling, average pooling, attention pooling. For node classification tasks, the output layer of the spectral GNN is often a fully connected layer.

4. Temporal Graph Neural Networks

Traditional Graph Neural Networks are limited in applications that work with time-varying data. These GNN models have difficulty modeling temporal dynamics because the graph structure they work with is static, neglecting the changing relationships that actually exist. For example, in social network analysis, a model must understand the changing social interactions over time in order to predict influential individuals or discover communities. Similarly, in financial applications like risk assessment of investment decisions, market dynamics must be taken into account to make accurate predictions. To overcome these limitations, Temporal Graph Neural Networks (TGNNs) have become a promising extension to traditional GNNs and a relevant area of research. The models allow the modeling of temporal dependencies in addition to the spatially arranged data. [7]

4.1. Temporal Graph Neural Network Architectures

In the past, researchers have proposed a number of TGNN architectures. In addition to the algorithmic classification of GNNs, namely spatial or spectral, there is another major design decision for TGNNs in the time variant method. The temporal information can be directly defined in the graph structure, or the GNN can be extended by operations handling the temporal information. A common approach are Hybrid Graph Neural Networks, where time-varying information is processed in another ML algorithm. [7]

Temporal information can be a static information: a time stamp, which can be modeled as a node feature, or the temporal distance between nodes, which can be modeled as an edge feature. An example for this is pandemic forecasting [8].

Temporal information related to a node or an edge can also be dynamic, it can be referred to as time varying information, e.g. time series information. In this case, GNN does not need to be extended yet, because the information can be modeled as multidimensional features. Time series information can also be passed as an argument into the message passing function, where GNN layers correspond to specific time steps [9]. In spectral GNNs, this kind of information can be used to set the parameters for convolution kernels [10].

If the graph is dynamic, i.e. links change over time, an extension of the GNN is inevitable. A dynamic graph representation can be a sequence of graph snapshots over time, where the individual graphs serve as input for a Graph Attention Network (GAT), to be later merged in an output function [11]. Another option is the temporal evolution of the graph, where in each layer the neighbor nodes are defined by the corresponding graph snapshot of the layer at a given time [12].

Hybrid GNNs for processing temporal information can be used, too. There is no standard way for information flow between time and graph modules. Nevertheless, the capabilities of specific architectures can be used to take advantage of the different ways of handling time. [7]

- 1D-CNN time module: A convolutional neural network with one-dimensional input is used to extract temporal patterns. The convolution operation allows the model to have a small number of parameters as well as translation invariance, so that patterns are detected regardless of their temporal position. In addition, manual modification can help the model to better fit the data. 1D-CNN can complement the GNN in a way that can be considered as sandwiched, where its input and output is the output and input for a GNN layer, respectively. [13]
- LSTM time module: Long Short-Term Memory is a type of Recurrent Neural Network (RNN). It allows the network to selectively remember

and forget information based on its relevance to the context. The component can model sequential patterns, such as trends, and seasonality. An application here is to predict PV power forecasting by modeling spatial and temporal dependencies between power plants. [14]

4.2. PyTorch Geometric Temporal

As machine learning algorithms evolve in research and real-world applications, software emerges that enables widespread use and rapid implementation of machine learning models. However, in the early research stages of new architectures, well-suited libraries often do not exist and implementation tasks are complicated. Motivated to create an open source machine learning algorithm that could handle non-static node features in graphs, Rozemberczki *et al.* came up with the PyTorch Geometric Temporal framework. [15]

Their goal was to create a user-friendly and functional software. For easy inspection of the runtime state, they designed the software in a modular way with limited number of public methods. Furthermore, the system provides test coverage, documentation, practical tutorials, continuous integration, package indexing, and frequent releases.

Providing this framework, the authors claim that PyTorch Geometric Temporal is the first deep learning library designed for neural spatiotemporal signal processing. Figure 1 shows the different scenarios of GNN's non-static properties that the framework is able to handle.

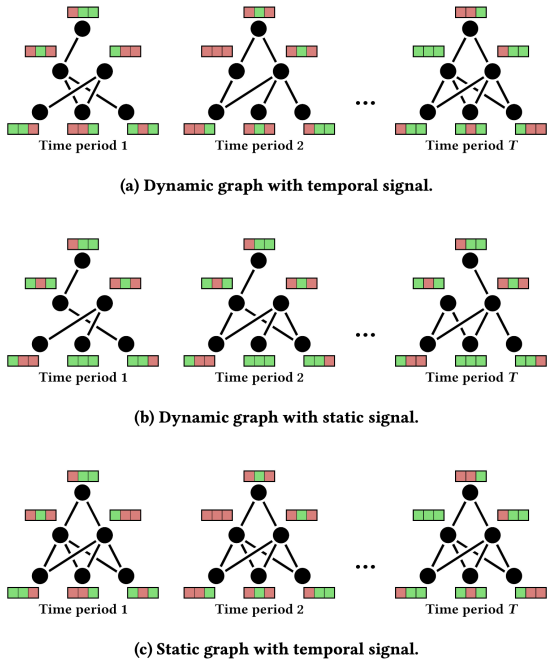


Figure 1: PyTorch scenarios [15]

Examining the framework's predictive performance, the authors find that PyTorch Geometric Temporal has similar predictive performance to recurrent neural networks on regression tasks. Among possible future directions, they mention considering continuous time or time differences between temporal snapshots that are not constant. Another possibility they see is the inclusion of temporal models that operate on curved spaces, like hyperbolic or spherical spaces.

4.3. Current Challenges in Temporal Graph Neural Network Development

While the potential of TGNNs is undeniable, there are still challenges that may slow down TGNN research.

A prominent problem is the lack of benchmarking capabilities. Models are trained for a specific problem, but due to limited standardized datasets and evaluation metrics, the models are not tested on datasets from different domains. This makes it uncertain whether they are suitable for other applications. [7]

A further difficulty for the rather new TGNNs is the need to adapt the learning methods to avoid oversmoothing. For GNNs, techniques such as dropout, virtual nodes, and neighbor sampling exist. Due to the variety and complexity of architectures, no solution can serve as a general solution [16]. Therefore, a lot of experiments need to be done on the even more complex TGNNs.

Another well-known problem is that institutions working with privacy-critical data are often unable to publish data. This is related to missing methods to create privacy-preserving representations. One solution is federated learning, a training method for dealing with data isolation between different sources. Guannan Lou *et al.* have demonstrated the effectiveness of a federated learning framework for a TGNN. [17]

5. Related Work

There is a range of applications where TGNNs show good performance.

Traffic prediction is a major domain for TGNN research. The temporal GNNs have superior performance over traditional GNNs in tasks like traffic planning and route planning. Their predictions show to be more accurate and better handle dynamic traffic conditions, such as traffic fluctuations and congestion. [18]

Pandemic forecasting for the TGNN became a big research domain during covid pandemic. It has shown to achieve state-of-art forecasting. [8]

PV production forecasting is of great importance for the transition to renewable energy. TGNNs have shown to outperform state-of-the-art methods for PV forecasting [14]. Therefore, TGNNs have great potential for long-term applications such as infrastructure planning as well as short-term applications such as efficient dispatching of other sources or informed decisions related to energy markets.

5.1. Internet Traffic Forecasting using Temporal-Topological Graph Convolutional Networks

A prospective application for TGNNs can be internet traffic forecasting. Being part of everydays life, it is important the internet works on fast and reliable infrastructure. With billions of connected devices and exponentially growing traffic it needs good solutions to master this challenge. However, it opens up opportunities for continuous innovation in network technology and supporting

 doi: 10.2313/NET-2023-11-1_20

algorithms. One advancement for infrastructure planning and network resource management lies in accurate internet traffic prediction.

Internet traffic prediction belongs to the class of time series forecasting problems. In this field linear prediction methods are used as well as neural networks, being capable to model non-linear data. However existing neural network algorithms in internet traffic forecasting often ignore network topology as they mainly model temporal data of traffic flow series. [19]

Zhenjie Yao *et al.* [19] proposed "Temporal-Topological Graph Convolutional Networks" (TTGCN), a TGNN architecture modeling the links' throughput of internet traffic in time series with also capturing network topology for predicting internet traffic. The graph representation looks as follows: A network link makes a node in the TTGCN graph, where edges exist, when the links are connected to the same router. The model processes in turn the temporal convolution and graph convolution, as illustrated in Figure 2. The temporal convolution extracts the temporal features while the graph convolution uses the new representation connecting it to the topological information. The temporal convolution is a gated linear unit that takes as input the feature matrix of the graph, initially the time series data of the links. Two matrices are generated by multiplying the input by the two different convolution kernels set in the current layer. A sigmoid function is applied to one of the matrices, which is then merged with the other matrix by computing the Hadamard product. The output is an updated graph feature matrix with one feature less. The new feature matrix is passed to the graph convolution layer that works with the graph's spectral domain. Here the researchers came up with two different approaches for representing the graphs adjacancy matrix. One is a normal adjacancy matrix for the graph as described before, the other is defined

$$\hat{A}_{i,j} = \begin{cases} B, & \text{if link } i \text{ heads to tail of link } j \\ -1, & \text{if link } i \text{ heads to tail of link } j \text{ and v.v.} \\ -1, & \text{if link } i \text{ heads to head of link } j \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

where the optimal parameter B is to be found by testing for the smallest error. Head and tail are the routers that a directed link passes traffic to and from respectively. The main operation here is the multiplication of its input with the graphs's Laplacian matrix and a matrix of learnable parameters. Following this structure, the final temporal convolution layer's output contains one remaining node feature. At this stage the feature matrix is passed to a fully connected layer, whose results are the predictions for every network link. Obtaining only one time step's predictions, the later time steps' predictions need to be obtained recursively.

The model's performance was tested with data from the UKERNA academic network backbone by Simple Network Management Protocol (SNMP). Traffic of 18 links connecting 8 core routers was observed for over a month. Samples for all links were taken every 10 minutes. In the paper the MAE and RMSE are compared for different prediction models, namely Historical Average, ARIMA, a popular linear model for time-series forecasting, Gated Recurrent Unit, Spatio-Temporal Graph Convolution Networks (STGCN), a TGNN showing good performance in road traffic prediction. Both the temporal GNNs were measured with the normal and the advanced adjecency matrix: TTGCN+, STGCN+. In the test the model should predict the next 9 time steps after being initialized with data of 12 time steps. The models were trained with data of 31 days and were tested on the data of the 8 remaining days. The test results show best performance for TTGCN+ then STGCN+, TTGCN, STGCN, pointing out the proposed model achieved the best prediction performance. The paper does not provide information if the best adjacancy matrix parameter was calculated for STGCN+, too. With TTGCN+ having a 13.7% lower RMSE than STGCN+ and over 10% lower RMSE than TTGCN+ with next higher parameter shows that having a good data representation is crucial for exploiting a prediction model's potential.

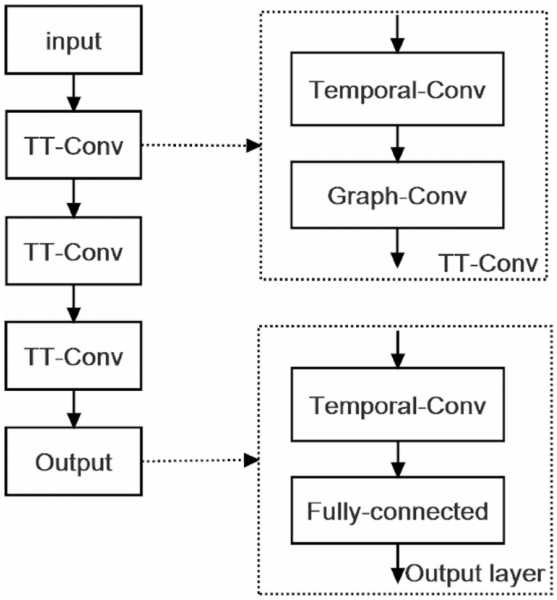


Figure 2: Architecture of TTGCN [19] ©2021 IEEE

6. Conclusion

In this paper, Temporal Graph Neural Networks are explained. It provides relevant information on machine learning algorithms and describes how common spectral and spatial GNNs are constructed. It can be concluded that TGNNs show good prediction performance on data with spatial and temporal relationships. Their architectures are usually combinations of common patterns in machine learning. One can expect that new TGNN architectures will emerge to leverage their capabilities in even more applications. However, there are challenges that need to be addressed. Contributions to benchmarking tools are needed, GNN methods against oversmoothing need to be adopted, further advances for federated learning must be driven.

References

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, 2018.

[2] H. Taud and J. Mas, *Multilayer Perceptron (MLP)*. Cham: Springer International Publishing, 2018, pp. 451–455. [Online]. Available: https://doi.org/10.1007/978-3-319-60801-3_27

[3] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," 2015.

[4] F. Günther and S. Fritsch, "Neuralnet: training of neural networks." *R J.*, vol. 2, no. 1, p. 30, 2010.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[6] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[7] Z. A. Sahili and M. Awad, "Spatio-Temporal Graph Neural Networks: A Survey," 2023.

[8] A. Kapoor, X. Ben, L. Liu, B. Perozzi, M. Barnes, M. Blais, and S. O'Banion, "Examining COVID-19 Forecasting using Spatio-Temporal Graph Neural Networks," 2020.

[9] L. Wang, A. Adiga, J. Chen, A. Sadilek, S. Venkatramanan, and M. Marathe, "CausalGNN: Causal-Based Graph Neural Networks for Spatio-Temporal Epidemic Forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, pp. 12 191–12 199, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/21479

[10] S. Hadou, C. I. Kanatsoulis, and A. Ribeiro, "Space-Time Graph Neural Networks," 2022.

[11] A. Fathy and K. Li, "Temporalgat: Attention-based dynamic graph representation learning," in *Advances in Knowledge Discovery and Data Mining*, H. W. Lauw, R. C.-W. Wong, A. Ntoulas, E.-P. Lim, S.-K. Ng, and S. J. Pan, Eds. Cham: Springer International Publishing, 2020, pp. 413–423.

[12] Y. Fan, M. Ju, C. Zhang, and Y. Ye, *Heterogeneous Temporal Graph Neural Network*, pp. 657–665. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611977172.74

[13] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, "Spatiotemporal Graph Neural Network for Performance Prediction of Photovoltaic Power Systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 17, pp. 15 323–15 330, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17799

[14] J. Simeunović, B. Schubnel, P.-J. Alet, and R. E. Carrillo, "Spatio-Temporal Graph Neural Networks for Multi-Site PV Power Forecasting," *IEEE Transactions on Sustainable Energy*, vol. 13, no. 2, pp. 1210–1220, 2022.

[15] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon, and R. Sarkar, "PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models," 2021.

[16] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini, "Graph Neural Networks for temporal graphs: State of the art, open challenges, and opportunities," 2023.

[17] G. Lou, Y. Liu, T. Zhang, and X. Zheng, "STFL: A Temporal-Spatial Federated Learning Framework for Graph Neural Networks," 2022.

[18] Y. Li, W. Zhao, and H. Fan, "A Spatio-Temporal Graph Neural Network Approach for Traffic Flow Prediction," *Mathematics*, vol. 10, no. 10, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/10/1754

[19] Z. Yao, Q. Xu, Y. Chen, Y. Tu, H. Zhang, and Y. Chen, "Internet Traffic Forecasting using Temporal-Topological Graph Convolutional Networks," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

Saving and Recovering Systems

Philipp Tekeser-Glasz, Sebastian Gallenmüller*, Manuel Simon*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: philipp.tekeser-glasz@tum.de, gallenmu@net.in.tum.de, simonm@net.in.tum.de

***Abstract*—The Chair of Network Architectures and Services operates multiple testbeds that allow researchers to develop and run reproducible network experiments. Reproducibility is achieved by running experiments on test nodes that have no persistent storage. Instead, test nodes boot a live system via PXE. This ensures that an experiment always starts with the same initial state. Users can reserve test nodes in advance using a calendar web interface. The facts that test nodes cannot store data persistently and that they are shared between users create a problem for users who are developing new experiments since all development progress is lost after a restart of the test node.**

This paper presents an approach to overcome this problem by using virtual machines that can be saved to a host with persistent storage and later restored on a test node.

***Index Terms*—virtualization, network experiments, testbeds, reproducibility, development**

1. Introduction

The testbeds at the Chair of Network Architectures and Services are managed by the Plain Orchestration Services (pos) which allows researchers to develop and run reproducible network experiments [1]. One of the goals of the testbeds and the pos framework is reproducibility, which means that different researchers are able to obtain the same result using the same experiment setup [2]. Each testbed consists of multiple test nodes that run the experiment code and a management node that controls the execution of the experiment. As a measure to achieve reproducibility, test nodes run live systems and do not have any persistent storage that could contain leftover data from previous users. This ensures a clean state at the beginning of an experiment.

However, during the development phase of an experiment, this architecture prevents users from saving the current state of a project between sessions since all data is lost after a reboot. This paper presents an approach to develop an experiment in virtual machines on a test node, which can be saved to the management node and restored later.

Section 2 gives a short introduction to architecture and the technologies used in the testbed. The problem of lost development progress that arises from the stateless architecture is described in Section 3 and a possible solution is explained. In Section 4, the implementation of that solution is presented. Section 5 shows an example workflow from the user's perspective. The limitations of the chosen approach are outlined in Section 6.

2. Background

This section explains the architecture of the testbed. The testbed consists of a management node and multiple test nodes. Test nodes run the experiment code. Some of them contain specialized network hardware for specific experiments. In order to guarantee a defined state at the beginning of an experiment, these nodes do not have any persistent storage that could contain leftover data from previous experiments. A live operating system is loaded over the network using the Preboot Execution Environment (PXE). Therefore, any changes to the system are lost after a reboot. In addition to the test nodes, each testbed has a management node to control the test nodes. On the management node, each user has a persistent home directory. Each test node contains a Baseboard Management Controller (BMC); a device that allows the management node to send commands to the test node even if the operating system is unresponsive or has not been started. The management node communicates with the BMCs over the network using the Intelligent Platform Management Interface (IPMI).

In order to ensure that users do not interfere with each other by accessing a node at the same time, pos offers a web interface through which nodes can be reserved in advance. Users can then log in to the management node using SSH and then use the pos command line tool to set up their test nodes.

Only the management node is accessible from the Internet. If a user wants to connect to a test node, they first have to connect to the management node via SSH and can then establish an SSH connection to the test node from there.

For development purposes, pos also offers the option to create multiple virtual machines on a physical test node. This is done by setting up the physical test node and launching virtual machines using the management software libvirt [3]. Virtual machines are managed and booted using the same protocols that physical test nodes use. In order to allow pos to control virtual machines, the host runs VirtualBMC [4] which accepts IPMI commands and passes them on to the virtual machines running under libvirt. The virtual machines receive their live operating systems via PXE and do not have any persistent storage either.

3. Problem

While the fact that test nodes do not store data persistently allows researchers to run automated and repro-

 doi: 10.2313/NET-2023-11-1_21

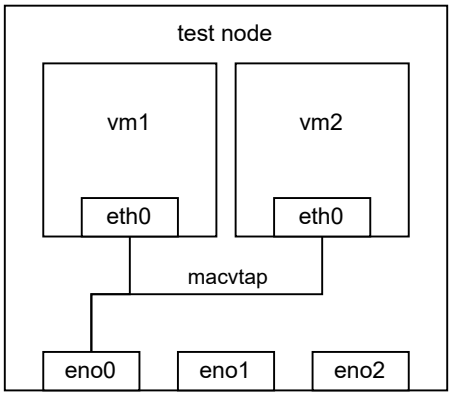


Figure 1: Networking on a test node

ducible network experiments, it is difficult to develop experiments in this environment since all files are lost after a reboot and users have to manually save and restore their progress between sessions.

This paper presents an approach to automatically save the state of virtual machines to the management node from where it can later be restored on another test node.

The chosen approach consists of creating memory images of the virtual machines on a test node, copying them to the management node where each user has a persistent home directory and later restoring the virtual machines.

Additionally, it is necessary to save the network configuration of the virtual machines. In a typical setup, there is a virtual management network that connects all virtual machines to the management interface of the host using a macvtap bridge device, as shown in Figure 1. The management interface is the one through which the node can communicate with the management node. This network allows pos to manage the virtual machines and users to connect to them via SSH from the management node. Other physical interfaces that are connected to other nodes might be present for experiments. For this reason, the correct interface has to be selected when setting up the virtual machines for the first time and after restoring since the new host could have other network interfaces than the old one. This selection is done automatically by searching the interface through which the default gateway is reachable.

To ensure that pos can find and control the virtual machines, the virtual network interfaces have to use MAC addresses that are derived from the host's IP address. When restoring the machines on a different test node, the MAC addresses have to be changed to match the new host's IP. This is done by removing the interface from the virtual machines before saving and adding a new interface with the updated MAC address after restoring. After that, the guest system has to be reconfigured to recognize the new interface with a different MAC address. Since there is no working network connection at this stage, the reconfiguration is performed using a virtual serial port. This reconfiguration of the management network allows restoring the virtual machines on any test node.

A simpler approach is to leave the network interfaces untouched, but this comes with the limitation that the virtual machines can only be restored on the test node from which they were saved because the MAC addresses will otherwise not match the pattern that is expected by pos and it will not be possible to establish a connection to the machine. The advantage of being able to freely choose a host is that users can work at any time without having to wait until a specific test node becomes available.

It is important to note that the chosen approach is not meant to replace the reproducible architecture of the testbed. While the approach simplifies the development process of new experiments, actual measurement results should only be acquired using the reproducible approach described in [1].

4. Implementation

The development workflow consists of three steps that will be explained in this section. For each step in the workflow, there is a Bash script that the user executes on the management node. The scripts connect to the specified test node to perform the necessary tasks. A simplified overview of the interactions between the nodes in each step is shown as a sequence diagram in Figure 2. Each of the following subsections corresponds to a frame in Figure 2.

4.1. Creating virtual machines

This process is based on the existing example code that is used to create virtual machines in the testbed [5]. In this step, the user starts a Bash script on the management node which starts the selected test node and installs the necessary virtualization software. After that, the virtual management network is created. Then a user-specified number of virtual machines is created. Each virtual machine has a network interface that is connected to the management network. The MAC address is calculated based on the IP address of the host and the ID of the virtual machine. For each virtual machine, a virtualBMC is started on the host that listens on a specific port based on the ID of the VM. Once this is done, pos can start the VMs, which then load their live operating system via PXE like physical machines. When the boot process is completed, getty is started on the virtual serial port in order to allow a reconfiguration of the network settings after restoring. Users can now establish SSH connections and start working on their projects.

4.2. Saving virtual machines

To save the virtual machines, a Bash script is executed on the management host which launches another Bash script on the test node that performs multiple steps. The first step is to create a file that contains a list of all virtual machines and their IDs. The second step is to remove the management interfaces from the virtual machines. If the user has configured other interfaces, the script will ignore those. In the third step, the definitions of the virtual machines are stored as XML files. These files contain general configuration data, e.g. memory size and serial ports. After that, the actual memory dump is created and compressed using gzip. The last step is to export the network configuration from libvirt and replace the host

 doi: 10.2313/NET-2023-11-1_21

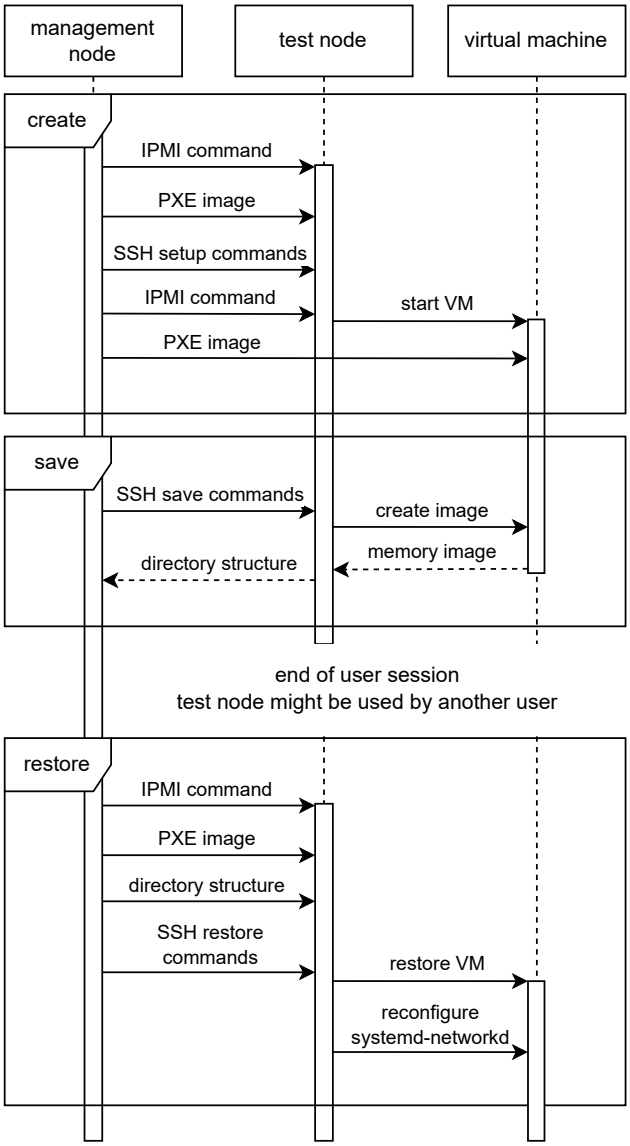


Figure 2: Simplified sequence diagram of the implementation

```
vm_save
|-- net
|   `-- net.xml
|-- vm
|   |-- vm1.gz
|   `-- vm2.gz
|-- vmmacs
`-- vmxml
    |-- vm1.xml
    `-- vm2.xml
```

Figure 3: Directory tree after saving

specific interface name with a placeholder variable that will be replaced on the new host when restoring. The final directory tree on the test node after all steps are completed is shown in Figure 3. This directory is then copied to the user's home directory on the management node using SCP. The compressed memory image of a Debian 11 system with no additional software installed has a size of 1 GB.

4.3. Restoring virtual machines

The restoration process begins like the creation process by starting the test node and installing the virtualization software. In addition to that, the saved directory is copied from the management node via SCP. First, the network configuration is read and the placeholder variable is replaced with the management interface. Then, the virtual machines are redefined using the XML files. After that, the memory images are decompressed and restored. Each virtual machine gets a new management interface with a MAC address based on the host's IP address and the ID that is read from the `vmmacs` file. However, the guest system does not recognize the new interface because of the changed MAC address. For this reason, an expect [6] script is launched to replace the MAC address in the interface configuration file of systemd-networkd on the guest. After this step, the virtual machines can be reached from other hosts on the network. In order to allow pos to send IPMI commands to the virtual machines, virtualBMC is started on the host system. This allows the user to reset the virtual machine to a clean state using the pos command line tool.

5. Example Workflow

This section shows the development workflow using the scripts. As an example, iperf3 will be installed on two virtual machines on test node `vmexp1`. These virtual machines will then be saved to the management node and restored on test node `vmexp0`. All shown commands are run on the management node.

First, two virtual machines with Debian 11 are created on the test node `vmexp1`.

```
./experiment.sh vmexp1
```

After the script has finished executing, there are two virtual machines called `vmexp1-vm1` and `vmexp1-vm2`.

It is now possible to establish an SSH connection and start developing. In this simple example, we will install iperf3 on both virtual machines.

```
ssh vmexp1-vm1 apt install iperf3
ssh vmexp1-vm2 apt install iperf3
```

Then, iperf3 is run in server mode on `vmexp1-vm1` and in client mode on `vmexp1-vm2`.

```
ssh vmexp1-vm1 iperf3 -s -D
ssh vmexp1-vm2 iperf3 -c vmexp1-vm1
```

This command will measure the throughput of the virtual network connection between the two virtual machines.

At the end of a session, the user can save all virtual machines to a new directory called `save_dir` using the following command.

```
./save_vms.sh vmexp1 save_dir
```

The directory `save_dir` will then contain the structure shown in Figure 3.

In order to start a new session on the node `vmexp0`, the following command is used.

```
./restore.sh vmexp0 save_dir
```

 doi: 10.2313/NET-2023-11-1_21

Now, the user can continue working on the virtual machines called `vmexp0-vm1` and `vmexp0-vm2`. In this case we can run iperf again in order to see that the installed package is present and that the network connection has been restored correctly. Note that the iperf server process is still running on `vmexp0-vm1` since we saved a full memory image of the virtual machines.

```
ssh vmexp0-vm2 iperf3 -c vmexp0-vm1
```

Like before, this command will show the throughput of the virtual network connection between the two restored virtual machines.

6. Evaluation

The chosen approach simplifies the development process of new experiments by allowing users to create backups of virtual machines and restore them later on a different host. It makes a more effective use of testbed resources possible since users only have to use one physical test node when developing experiments that would normally require multiple nodes.

However, there are some limitations. First, it is only possible to restore virtual machines on a different host when there is only a management network that has the structure shown in Figure 1. Other interfaces are ignored by the script and might therefore lead to an error when restoring on a new host where the interface is not available. More complex network setups, e.g. passthrough interfaces, are possible but restoring will only work on the same host where these interfaces are available.

When restoring network interfaces of type virtio on the new host, AppArmor caused an error. For this reason, AppArmor had to be completely disabled via a kernel parameter in order to be able to restore these network interfaces.

Another limitation is that the network reconfiguration on the guest system is currently only supported for systemd-networkd. It is therefore only possible to use specific operating systems in virtual machines. In this case, only Debian 11 was tested, which is commonly used on the testbed.

When an experiment is ready to run, it might be necessary to make changes to the code in order to run it on physical hardware.

7. Conclusion

We explained the stateless nature of nodes in the testbed and the advantages it has for reproducible network experiments. This created the problem of losing progress during the development phase of an experiment. We proposed a solution to this problem based on virtual machines which can be saved to the management node and restored on a different test node. An implementation of this approach using Bash scripts was presented. We also provided a demonstration of the implementation in which we showed the steps to create, save and restore a session.

In the future, this solution could be integrated into the pos command line interface in order to simplify the development workflow. Another topic for future work could include the development of a more complex example project using the presented approach.

References

[1] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos framework: A methodology and toolchain for reproducible network experiments," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 259–266. [Online]. Available: https://doi.org/10.1145/3485983.3494841

[2] ACM, "Artifact Review and Badging - Current," https://www.acm.org/publications/policies/artifact-review-and-badging-current, 2020, [Online; accessed 18-June-2023].

[3] "libvirt: The virtualization API," [Online; accessed 18-June-2023]. [Online]. Available: https://libvirt.org/

[4] "How to use VirtualBMC — virtualbmc 3.0.2.dev5 documentation," [Online; accessed 18-June-2023]. [Online]. Available: https://docs.openstack.org/virtualbmc/latest/user/index.html

[5] S. Gallenmüller, "vm-example," https://gitlab.lrz.de/I8-testbeds/pos-examples/-/tree/master/tutorials/vm-example/bullseye, 2021, [Online; accessed 18-June-2023].

[6] D. Libes, "expect: Scripts for controlling interactive processes." *Computing Systems*, vol. 4, pp. 99–125, 03 1991.

Network Insights with P4 In-Band Network Telemetry

Sebastian Warter, Sebastian Gallenmüller*, Kilian Holzinger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: sebastian.warter@tum.de, gallenmu@net.in.tum.de, holzinger@net.in.tum.de

***Abstract*—Understanding what happens to packets in layer 2 networks is inherently difficult. The transparent nature of network switches can make the identification of faulty network components a time-consuming search. The P4 In-Band Network Telemetry (INT) can help by aggregating switch telemetry information in the data packets. This paper explains the concepts of INT in a simple use case where we want to identify a high-latency link. With the P4 implementation from the GÉANT project, we demonstrate this scenario in a virtual network. The evaluation of this experiment shows that INT is suitable for the use case but has some overhead in the emulated network.**

***Index Terms*—P4, in-band network telemetry, INT, monitoring**

1. Introduction

The operation and administration of large networks can be a challenging task. For example, assume that we are dealing with a video conference company network. Such a network usually has to transmit a considerable amount of UDP packets with low latency. There is likely also a monitoring system in place that can detect abnormal high latencies or low data rates from an end-to-end perspective.

However, this information does not help to identify the problematic network component. If the network consists of layer 3 routers, a tool like `tracepath` can help to narrow the issue down. Unfortunately, it does not work in all cases. In complex networks, the routers might handle the echo request differently than the real traffic without triggering the problem. The problem might also involve a layer 2 switch or a link between two switches. They are transparent to the `tracepath` tool.

Ideally, we would have a technology that can track on-demand how regular data packets move through the switches in a layer 2 network. If a switch sends metadata to a monitoring system every time it encounters a tracked packet, we could reconstruct the layer 2 path of each packet including time information. This approach would still require a pre-existing identifier in the packets, which is not always available.

A different approach is to add the metadata to the packet itself at each switch it passes. The aggregated metadata then has to be removed from the packet once it leaves the network. This does not require unique IDs for every packet.

The second approach can be realized using P4 programmable switches and the In-Band Network Telemetry (INT) specification described in Section 2. The following Section 3 describes how to solve the use case described at the beginning with INT. In Section 4, we pick a suitable P4 implementation to create a test setup in Section 5. This test setup is then evaluated in Section 6 on our use case.

Similar demonstration setups were also described in related work. For example, Kim et al. briefly described a simple demonstration setup based on an older INT specification [1]. Parniewicz et al. described their results on more complex network environments [2]. This paper focuses on a simple use case instead.

2. Background

Traditionally, professional network hardware is highly specialized and has limited configuration possibilities. Even though this usually means it operates efficiently, it has the downside that new functionality or protocols usually require buying expensive new hardware. The P4 ecosystem presented in Subsection 2.1 aims to change that with a programming language for packet processing. In order to emulate networks with P4 switches, the mininet project described in Subsection 2.2 can be used. The flexibility of P4 also allows to implement more advanced monitoring systems like the In-band Network Telemetry (INT) in Subsection 2.3.

2.1. P4

P4 stands for "Programming Protocol-Independent Packet Processors" and was first described by Bosshart et al. [3]. It is a programming language for packet processing that describes detailed steps to perform on incoming network packets. It does not assume the usage of standardized protocols like IP or TCP, which is usually a prerequisite for traditional network hardware. Instead, it allows the programmer to define the header structures themselves. Due to this flexibility, existing P4-based hardware can also be used for network protocols that do not exist yet. [3].

The P4 language (more precisely, the 2016 revision $P4_{16}$) has two essential language constructs. "Parsers" parse the headers of an incoming packet. They use a programmer-defined state machine to identify and parse nested headers. "Control blocks" are imperative programs. They can use tables to trigger programmer-defined actions based on the value of header fields. The table entries are usually not part of the program and are configured by the control plane at startup or runtime (for example, a routing table). [4].

The possible actions a switch can perform, and the processing pipeline itself, is not enforced by P4. Instead,

 doi: 10.2313/NET-2023-11-1_22

it provides the syntax to describe the capabilities of switch architectures. For example, a certain switch model might provide functions to calculate CRC checksums, while other switch models do not support them. [4].

The reference switch architecture v1model[1] is based on a simple packet processing pipeline with six predefined steps (including ingress/egress pipeline and emitting headers at the end). The P4 project also provides a software implementation of a switch called "Behavior Model 2" ("bmv2") that supports the v1model [5].

2.2. Mininet

Developing P4 applications using real hardware is difficult. The hardware is usually expensive and difficult to reset to a clean state or to debug. For developing systems with P4-based switches, a virtual network setup is more convenient. Although such an emulated network fails to reflect real networks accurately, it can be a valuable tool for network experiments [6].

A popular tool for creating virtual networks is mininet. It is based on Linux's built-in virtualization capabilities for network interfaces. Similar to container solutions like Docker, each virtual host is represented by a process with virtual network interfaces. This way, hosts in a mininet network can efficiently run normal applications. Switches in a mininet network are OpenFlow-compatible software implementations. [7].

The mininet command-line interface directly supports the creation of simple, pre-defined network topologies. Once it is running, it offers commands to inspect the topology and run normal shell commands on the virtual hosts. This way, the network settings of hosts can be configured using normal Linux commands. If more complex topologies are required, they can be defined using an object-oriented Python API. [7].

In order to test P4 applications in mininet, the OpenFlow-based switch can be replaced with a bmv2 switch. The P4 ecosystem offers the tool `p4app` for this purpose. It is a convenient tool that can compile a P4 program, start a mininet topology, and configure the tables on the switches. Because it is based on docker containers, it offers a development environment with all necessary tools without using resource-hungry alternatives like virtual machines. [8].

2.3. P4 In-band Network Telemetry (INT)

Understanding what happens in layer 2 networks is inherently difficult. The switches in the network are, by design, supposed to be transparent to network traffic. This means in practice, that it is not directly possible to know which path a specific packet took through a network or how individual links affect the total latency. With sophisticated hardware, it is usually possible to access additional information like packet rates. Unfortunately, they only allow us to guess what is happening in the network.

To improve insights into networks, the P4 working group specified a protocol called "In-band network telemetry (INT)". It is designed to track the visited switches and their states of any packet flow in a P4-based network. For this purpose, INT allows storing the instruction to collect telemetry and the switch states in the already existing data packets. [9].

1. https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4

TABLE 1: INT Hop-by-hop header, adapted from [9]

0	1	2	3
Version | Flags	Reserved	Hop ML	Remaining Cnt
Instruction Bitmap	Reserved		
Last hop INT metadata			
...			
First hop INT metadata			

In INT 1.0, the instruction to collect telemetry originates from an "INT source" switch. It injects the header in Table 1 into the forwarded packets. In particular, it sets bits in the "Instruction Bitmap" which correspond to the information that should be collected. This switch and all "INT transit hop" switches can append their own state to this header while forwarding the packet. Eventually, an "INT sink" switch removes the header and sends the collected data to a monitoring system. The report format is not specified in INT 1.0. [9]

The collected metadata usually includes information like device identifiers or timestamps. This information can then be used to visualize the layer 2 path of packets. It also allows to calculate the link and switch latencies from timestamps in the metadata.

The previously described setup is called "INT-MD" in the most recent version 2.1 of INT. The new version also adds two additional modes. In "INT-MX" mode, the telemetry is sent directly to the monitoring system by each switch instead of appending it to the header. This avoids packets that grow too large and exceed the MTU. The "INT-XD" mode works similarly, but the switches do not create an INT header and use built-in instructions to send telemetry. [10].

3. A Simple Monitoring Use Case

In order to illustrate the potential of INT, this paper demonstrates its benefits in a simple use case. We assume that a network operator wants to localize latency issues in a layer 2 network. With traditional tooling, this would be difficult because layer 2 switches are usually transparent to network traffic.

This problem can be solved with a simple INT setup. In this setup, each monitored packet aggregates a history of switch states as it passes the network. When the packet leaves the network, this information is sent to a monitoring system. The monitoring system can then be used to analyze the data. In our use case, it can calculate latencies for each link based on the INT timestamp differences.

To demonstrate this in practice, we create a virtual test network. It uses

- mininet to create three bmv2 switches and two hosts
- P4 code to create and process the INT headers
- a simple INT-MD configuration with one INT source, one INT transit hop, and one INT sink

 doi: 10.2313/NET-2023-11-1_22

TABLE 2: INT switch implementations

Name	bmv2	INT version (mode)	working documentation
joshi	×	1.0/2.1 (all)	not tested
ONOS	✓	1.0 (MD)	×
GÉANT	✓	1.0 (MD)	✓

TABLE 3: INT collector implementations

Name	Backend	Works on Linux 5.19
INTCollector	InfluxDB, Prometheus	×
GÉANT	InfluxDB	✓

- node ids, ingress timestamps, and egress timestamps, and
- a monitoring system that can collect and visualize the INT data.

For a real INT deployment, it is usually desirable to collect additional data like congestion indicators. Even though they can provide valuable information to diagnose network issues, they are omitted here to avoid additional complexity.

4. P4 INT Implementations

Over time, many developers have implemented different versions of the INT specification in P4-based projects. This section briefly compares three well-documented implementations in Subsection 4.1. After that, it describes details about the GÉANT implementation in Subsection 4.2, which is used in the remaining sections of this paper.

4.1. Choosing a Suitable Implementation

In practice, we use two main software components for our test setup:

- a P4 implementation of a network switch with INT support, and
- an application that collects the INT headers and transforms them into a format compatible with an existing database system.

This paper considers and compares the three P4 implementations summarized in Table 2 and the two collectors summarized in Table 3. Numerous other implementations exist, but many are based on the outdated version 0.4 of INT or are not properly documented.

The implementation created by Joshi in [11] is one of the most recent ones and supports the latest INT version 2.1. Unfortunately, it is built solely for Intel Tofino hardware and does not support the bmv2. The INT 1.0 implementation in the Open Network Operating System (ONOS) uses the bmv2, but its documentation[2] is outdated and does not work in current versions of ONOS. The last implementation described by Parniewicz et al. [2] as part of a GÉANT project is similar but has a working documentation.

2. https://wiki.onosproject.org/display/ONOS/In-band+Network+Telemetry+(INT)+with+ONOS+and+P4

A frequently used collector is the INTCollector described by Tu et al. [12], which can send data to InfluxDB or Prometheus backends. Unfortunately, this implementation fails to start on current Linux versions. For demonstration purposes, we can also use the slower Python implementation included in the GÉANT project to store the data in InfluxDB.

Only the switch implementation from the GÉANT projects seems suitable for our virtual test setup. We use it in the following sections for our use case from Section 3. For simplicity, we also use the INT collector included in the GÉANT project.

4.2. The implementation of the GÉANT project

This P4 implementation of INT was created as part of a GÉANT project about network monitoring. It includes an implementation of INT 0.4 and 1.0 for both virtual bmv2 switches and Intel Tofino switches. It also has extensive documentation and also provides visualization tools. [13].

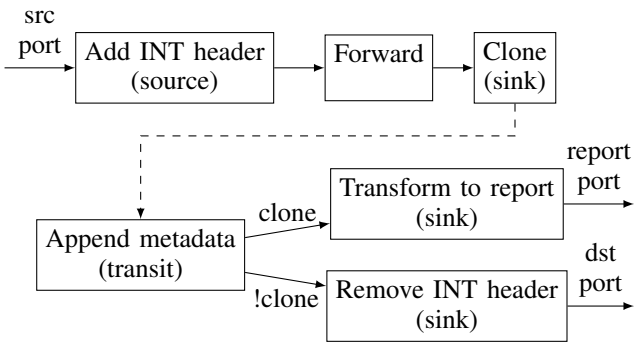


Figure 1: GÉANT P4 pipeline based on `int.p4`

The used P4 processing pipeline is visualized in Figure 1. It is configured using P4 tables [14].

In the ingress pipeline, the switch first adds the INT header if a packet should be monitored. This decision is made based on a flag to use a port as an INT source and a list of layer 3/4 endpoint addresses to monitor. Next, the egress port is picked from a static layer 2 address forward table. If it is a port configured as an INT sink, the packet is also cloned to the reporting port. [14].

The egress pipeline first appends the local metadata if there is already an INT header in the packet. If the destination is an INT sink port, the INT headers are removed in the next step. If it is a cloned packet sent to the reporting port, the packet headers are wrapped with a report header in order to send them to the IP address of the collector. The GÉANT project seems to use the same headers as the Telemetry Report 1.0 specification for this purpose [15]. [14].

5. Creating the P4 INT Test Setup

In the P4 INT demonstration setup, we have to install, configure, and start multiple software components for monitoring and the virtual network. For our simple requirements, the docker-based configurations shipped with the GÉANT project allow us to create an environment that matches our requirements:

1) Follow the instructions[3] to start and configure docker containers for InfluxDB and the grafana dashboard. These components act as the monitoring system that stores and visualizes the INT metrics. The collector itself is part of the next step.
2) Start the INT 1.0 mininet testbed that is shipped with the P4 implementation of the GÉANT project. The included instructions[4] describe how to start the collector and use `p4app` to create a virtual network in a docker container (external connectivity is not required). Note that the InfluxDB IP address should be a public IP of the host.

In the default configuration, the setup consists of three statically configured switches. Each can act as an INT source, transit hop, or sink. The network parts relevant to this paper are visualized in Figure 2 based on the `dump/net` output of mininet [7].

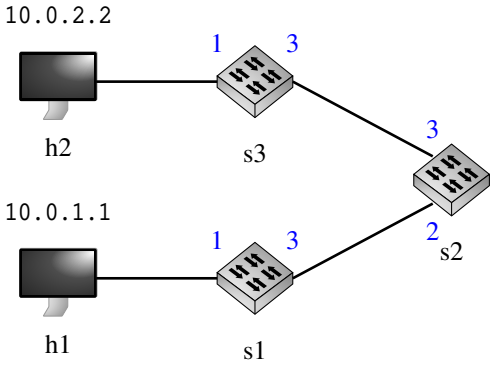


Figure 2: Layer 2 path between h1 and h2 with IP addresses and port numbers

This setup now includes all parts required for our monitoring use case from Section 3. For example, if we send data from host 1 to host 2, switch 1 acts as an INT source and adds instructions to collect all INT fields (due to the configuration in `commands1.txt`). All switches then add INT headers with telemetry while forwarding the packets. Switch 3 removes the headers and sends the INT data to the collector, which processes them and sends the telemetry to the InfluxDB server.

6. Evaluation of the Test Setup

We now use the previously created test setup to demonstrate how INT can help to localize latency issues. For this purpose, we first introduce a 5000 ms delay for data sent from `s1` to `s2`. Next, we start sending packets from `h1` to the IP address of `h2`. The GÉANT project provides us with the Python script `h1_h2_udp_flow.py` for this purpose. Both steps can be achieved by executing the commands in Figure 3 in the mininet prompt [7], [16].

The collected INT metadata can be analyzed in the Grafana dashboard. For our use case, we want to find

3. https://github.com/GEANT-DataPlaneProgramming/int-analytics
4. https://github.com/GEANT-DataPlaneProgramming/int-platforms/tree/master/platforms/bmv2-mininet

```
s1 tc qdisc add dev s1-eth3 root netem delay 5s
h1 python /tmp/host/h1_h2_udp_flow.py
```

Figure 3: Mininet commands used for our test setup

latency issues. The interesting values for this purpose are the pre-hop link delays (see Figure 4).

In our experiment, the delay between Switch 1 and 2 is about 6 s. This is higher than the near-zero delay between Switch 2 and 3. Therefore, we have identified our high-latency link.

Unfortunately, this experiment also reveals some limitations. There is an additional delay of about 1 s on the link `s1-s2` and not on the link `s2-s3`. We suspect that it is caused by the overhead of the software switch, but this hypothesis cannot be verified without tests on real hardware.

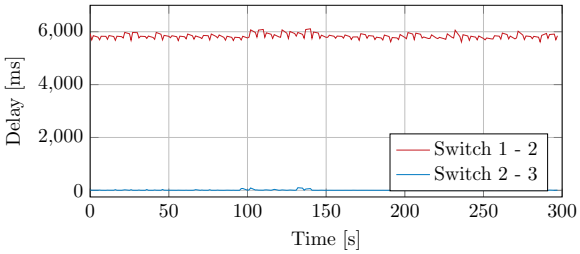


Figure 4: Link delays from experiment

Overall, this virtual experiment showed how it is possible to find high-latency links in an INT-capable network. Unfortunately, there is a significant overhead which would make it difficult to measure lower, more realistic latencies. Further evaluation of INT's accuracy in this use case would likely require P4-capable hardware and is out of the scope of this paper.

7. Conclusion

In this paper, we saw how INT can help to solve network problems in previously not possible ways. Based on a simple use case where we located latency issues in layer 2 networks, we explained INT and created a virtual demonstration setup. Our evaluation showed that INT is suitable for the use case but, at least in our emulated network, has a significant overhead.

Based on this setup, it is also possible to collect other potentially helpful INT data. For example, a network operator can decide to collect the queue occupancy or the exact layer 2 path of a packet. Accurately evaluating the precision of INT in these more complex use cases will require real hardware and future work. Such future work should also consider the newer version 2.1 of INT, which can provide additional possibilities to debug network issues.

References

[1] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[2] D. Parniewicz, T. Martínek, F. Pederzolli, D. Ding, M. Campanella, I. Golub, and T. Chown, "In-Band Network Telemetry Tests in NREN Networks," GÉANT Association, Tech. Rep., 2021.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014.

[4] The P4 Language Consortium, "P4_16 Language Specification," https://p4.org/p4-spec/docs/P4-16-v1.2.4.pdf, 2023, [Online; accessed 15-June-2023].

[5] P4 Project, "The Reference P4 Software Switch," https://github.com/p4lang/behavioral-model, 2023, [Online; accessed 15-June-2023].

[6] R. Oliveira, C. Schweitzer, A. Shinoda, and L. Prete, "Using mininet for emulation and prototyping software-defined networks," 06 2014, pp. 1–6.

[7] Mininet Project Contributors, "Mininet Walkthrough," http://mininet.org/walkthrough/, 2022, [Online; accessed 15-June-2023].

[8] P4 Project, "P4app," https://github.com/p4lang/p4app, 2019, [Online; accessed 15-June-2023].

[9] The P4.org Applications Working Group, "In-band Network Telemetry (INT) Dataplane Specification - Version 1.0," https://p4.org/p4-spec/docs/INT_v1_0.pdf, 2018, [Online; accessed 15-June-2023].

[10] ——, "In-band Network Telemetry (INT) Dataplane Specification - Version 2.1," https://p4.org/p4-spec/docs/INT_v2_1.pdf, 2020, [Online; accessed 15-June-2023].

[11] M. Joshi, "Implementation and Evaluation of In-Band Network Telemetry in P4," Master's thesis, KTH Royal Institute of Technology, 2021.

[12] N. V. Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "Intcollector: A high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*, 2018, pp. 10–18.

[13] D. Parniewicz, "Common P4-based INT implementation for bmv2-mininet and Tofino platforms," https://github.com/GEANT-DataPlaneProgramming/int-platforms, 2021, [Online; accessed 15-June-2023].

[14] ——, "INT Configuration Guide," https://github.com/GEANT-DataPlaneProgramming/int-platforms/blob/master/docs/configuration.md, 2021, [Online; accessed 15-June-2023].

[15] The P4.org Applications Working Group, "Telemetry Report Format Specification - Version 1.0," https://raw.githubusercontent.com/p4lang/p4-applications/master/docs/telemetry_report_v1_0.pdf, 2018, [Online; accessed 15-June-2023].

[16] F. Ludovici and H. P. Pfeifer, *tc-netem(8) Linux Manual Page*, 2011.

ISBN 978-3-937201-78-8
DOI 10.2313/NET-2023-11-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)