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 2022

March 11, 2022 – August 21, 2022

Munich, Germany

Editors

Georg Carle, Stephan Günther, Benedikt Jaeger

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 2022

Munich, March 11, 2022 – August 21, 2022

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET 2022-11-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Summer Semester 2022

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/`

Cataloging-in-Publication Data

Seminar IITM SS 22
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, March 11, 2022 – August 21, 2022
ISBN: 978-3-937201-76-4

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2022-11-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2022-11-1
Series Editor: Georg Carle, Technical University of Munich, Germany
© 2022, 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 2022. 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 Leon Kist with the paper *Survey on Scheduling Approaches in TSN* and Daniel Petri Rocha with the paper *Secure Data Marketplaces* .

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, November 2022

Georg Carle

Stephan Günther

Benedikt Jaeger

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

Advisors

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

Christopher Harth-Kitzerow (christopher.harth-kitzerow@outlook.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

Holger Kinkelin (kinkelin@net.in.tum.de)
Technical University of Munich

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

Christoph Schwarzenberg (schwarzenberg@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

Seminar Homepage

`https://net.in.tum.de/teaching/ss22/seminars/`

Contents

Block Seminar

Seminar

Performance Limitations of the QUIC Protocol

Jázmin Dojcsák, Benedikt Jaeger*, Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge96voh@mytum.de, jaeger@net.in.tum.de

***Abstract*—QUIC is a departure from the traditional TCP-based communication protocols as it is based on UDP. It implements solutions for many issues of modern networking, from head-of-line blocking to limiting ossification by middleboxes. It offers a full featured authenticated and encrypted end-to-end communication form, with low overhead, implemented in user-space. However, this does not come without costs.**

This paper examines the potential limitations of the new protocol and possible solutions to overcome them. In order to help understand the performance issues and the ongoing efforts to solve them, this paper gives historical background information on the developement of QUIC and highlights relevant characteristics of QUIC. The paper also reviews some open source implementations of QUIC.

The main goal of this paper is to review literature on this topic and summarize key findings on bottlenecks.

***Index Terms*—quic, performance**

1. Introduction

A new transport protocol recently standardized by the IETF offers a solution for challenges of today's ever-increasing internet traffic and for the slowdown of the computing performance growth.

Although QUIC is now a proposed standard by IETF, the original development of this protocol was driven by Google to provide an option for serving the rapidly increasing HTTPS traffic. Google had the advantage of being the vendor of a major web browser and additionally hosting popular web sites, so the company has implemented and tuned the protocol on an Internet-scale experimentation framework in the 2010s. This experiment was so wide scale that in 2016 30% of Google's traffic was served via QUIC, which was ca. 7% of the global Internet traffic [1].

Since then, QUIC gained wide acceptance, most desktop and mobile operating systems, the major browser vendors, the leading cloud providers and CDNs support QUIC. A relevant portion of Internet is served using this protocol, almost 25% of all websites, including the top 3 (Google.com, Youtube.com, Facebook.com), are driven by QUIC [2] [3].

This paper gives a brief overview about the motivation and the technical considerations of its development, describes the basic features of the protocol, introduces different implementations and presents challenges and potential answers to them. Chapter 2 describes the related protocols and problems that lead to the developement of QUIC. This chapter introduces the technical details, relevant for understanding the performance limitations presented in chapter 4. Chapter 3 reviews the most popular open source implementations of QUIC. Then, chapter 4 summarizes bottlenecks and possible efforts to overcome them. Finally, chapter 5 draws a conclusion.

2. Background

The Internet was built on mature and stable foundations like the TCP/IP stack, which slowly became limitation of this growth. QUIC was not the first attempt to resolve these issues, e.g. the transport protocol SCTP failed to gain wide acceptance [1].

HTTP Drawbacks. The application layer protocol HTTP was driving the Internet from the 90s, HTTP/1.1 has been in use since 1999. The limitations of the single request-response based nature of this protocol were getting more significant as web sites were getting more and more complex, causing hundreds of requests to load a single web page with its dependencies [4].

This is not the only problem with HTTP, among others, it suffers from head-of-line blocking and an HTTP client may require more than one TCP connection to effectively fetch a single web page [1].

Introducing HTTP/2. Serious efforts were made to overcome these limitations, Google developed SPDY and based on this, the IETF standardized the new HTTP/2 protocol in 2015. HTTP/2 offers multiplexing requests and responses, allows prioritization and features a more effective data framing. A HTTP/2 server may push content to clients before it is even requested. It has had considerable benefits, it eliminated the head-of-line blocking of HTTP/1.1 but it still suffered from the limitations of the underlaying transport protocol [5].

TCP Drawbacks. HTTP uses the connection-oriented TCP as transport layer protocol, providing a reliable data channel between the communicating parties. HTTPS adds a secure layer to the transmission (TLS). TCP requires a 3 way handshake to establish a connection, the secure layer needs 2 additional round trips to negotiate the secured channel [1]. On the other hand, TCP is a highly optimized and mature solution with an established ecosystem. It is implemented in the kernel of operating systems and widely supported by middlebox vendors.

doi: 10.2313/NET-2022-11-1_01

In addition to the handshake delay, retransmission of a multiplexed HTTP/2 request results in a head-of-line blocking delay [1].

2.1. Introducing QUIC

Although the page load time was enhanced by HTTP/2, the efforts to reduce web latency were less successful due to the nature of the underlaying TLS/TCP [1]. To overcome these limitations, Google started the development of a UDP based transport protocol, QUIC was born. Not only the transport layer was a novelty in this development: the QUIC stack was intentionally implemented in user-space to allow easier deployment and updates.

QUIC Features. One of the biggest accomplishments of the QUIC protocol is the 0-RTT handshake. QUIC does not have to perform a separate cryptographic and transport handshake [1]. If a connection was previously established between client and server, the client can start sending data with no additional round trips. Without former acquaintance, the first connection to a server has to be set up in 1-RTT. Figure 1 depicts QUIC's initial 1-RTT and Figure 2 shows a subsequent 0-RTT handshake:

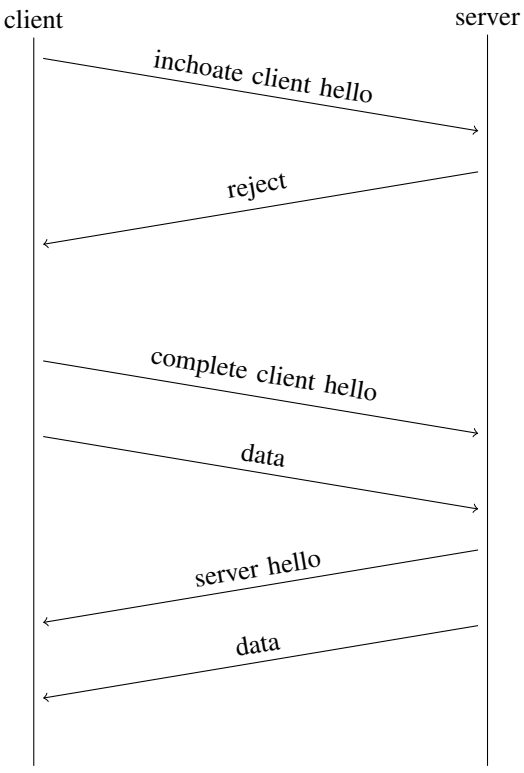


Figure 1: QUIC 1-RTT handshake [1]

1-RTT Handshake. : If a client is establishing a connection to a given server for the first time then it sends an "inchoate client hello" message which will be responded with a REJ message containing the server config with the necessary cryptographic data, including long-term Diffie-Hellman public value, the certificate chain, a signature and a source-address token [6]. The client persists this information for future use and it continues with the 0-RTT Handshake method.

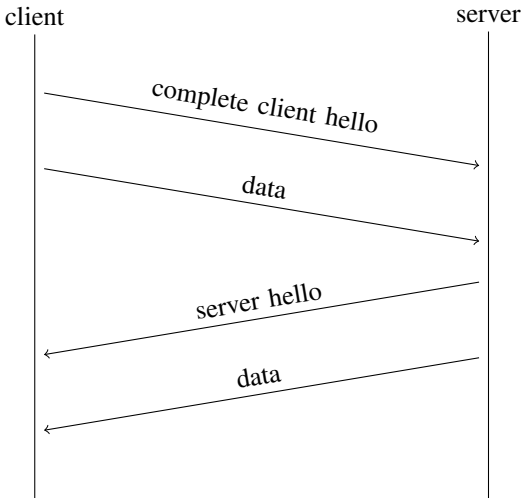


Figure 2: QUIC 0-RTT handshake [1]

0-RTT Handshake. : If the client possesses a valid server config then it can start (or continue) the communication with a "complete client hello" message with the cryptographic parameters of the client and without waiting for the response it may immediately start sending packets by applying an initial key calculated from the long term public value of server.

The server responds with a "server hello" message, including information for the session key, and any subsequent communication will be conducted using this ephemeral key, providing forward-secure encryption [1].

QUIC achieves this reduced connection setup, while guaranteeing **strong security** like TLS/TCP. Even in the case of 0-RTT, the communicating parties can negotiate the cryptographic parameters and the protocol version. The packet header and payload are authenticated and encrypted [12].

The packets are sent via UDP between the endpoints and multiple packets may be coalesced into a single UDP datagram. The packet payload contains an array of frames allowing **multiplexing** multiple logical streams in a single communication unit [12].

UDP may provide better latency than TCP, however it has no congestion control. Thus, QUIC has to implement **loss detection** and **congestion control** itself. QUIC uses generic congestion control signals allowing different algorithms to be applied, like the TCP CUBIC. Although congestion control is based on TCP's loss detection and congestion control, QUIC adds important enhancements to allow more efficient functionality like solving the retransmission ambiguity issue, by using unique packet numbers instead of sequence numbers [13].

QUIC makes it possible to avoid one of the most common bottlenecks of TCP, head of line blocking. A single QUIC connection establishes multiple streams, therefore, allowing out-of-order delivery. Hereby, lost packets only influence the stream in which they are received. Other streams remain unaffected and packet flow can continue. Each stream receives its own unique ID, streams started by the client will use odd numbers and even numbers will be assigned to streams initiated by the server [1], [14], [15].

doi: 10.2313/NET-2022-11-1_01

	QUANT	**picoquic**	**quicly**	**mvfst**
Vendor	NTAP	private-octopus	h2o	facebookincubator
Language	C	C	C	C++
TLS 1.3	picotls	picotls	picotls	fizz
Build system	cmake	cmake	cmake	cmake
Target	Linux, FreeBSD, macOS	Linux, FreeBSD, macOS, Windows	Linux, FreeBSD, macOS	Linux, macOS
Roles	client, library, server	library and test tools, test client, test server	client and server	client, server, library
GitHub Stars	211	321	495	1080
GitHub Forks	28	88	90	160
GitHub Commits	3329	3900	2079	4868

TABLE 1: Implementation details including GitHub metrics as of 2022-04-02 [7] [8] [9] [10] [11]

QUIC introduces a way to deal with network changes more efficiently than TCP. While TCP connections are identified by IP four tuples, QUIC ones are recognized by connection IDs. When a client's IP address changes, a TCP connection is automatically broken. However, with QUIC using connection IDs instead of IP four tuples, communication between client and server can continue. This effect is beneficial, when users are forced to change their network connections, e.g. when switching from mobile network to Wi-Fi [15].

HTTP/3. The pending Internet Draft defines HTTP/3 as an extension of HTTP/2. It replaces TLS/TCP with QUIC, which "provides protocol negotiation, stream-based multiplexing and flow control" [16]. HTTP/3 also offers a more efficient compression method [16].

3. Implementations

The first major QUIC implementation was part of the Chromium project by Google, it is now extracted into the project QUICHE, which powers Google's client products, like Chrome and the servers running in Google data centers [1]. Microsoft also develops an open source and cross platform implementation of QUIC, MsQuic is optimized for high performance offered in client products and on Azure [7]. Apple also included QUIC support in its latest mobile and desktop operating systems [17].

In this section some of the most popular open source implementations of the QUIC protocol are reviewed. Table 1 summarizes properties of the selected implementations.

Quant. The open source project QUANT (QUIC Userspace Accelerated Network Transfer) provides a client and server C implementation for POSIX and IoT platforms [7] [8].

The project is managed by the company NetApp Inc., is actively maintained and it complies with the latest versions of the standard (Draft-34 and v1 as of 2022-04-02 [7]). Quant supports a memory mapped network I/O solution (called "netmap") which makes it possible to bypass the traditional kernel network interfaces and thus allows very fast packet I/O. It uses picotls [18] for TLS 1.3 cryptography [8] [19].

mvfst. Mvfst (move fast) is a C++ QUIC library for servers and clients by Facebook. mvfst is intended to be performant on both the client and server side, it has been proven on mobile devices and in large data centers [11]. The software is deployed extensively in the Facebook server infrastructure, Instagram servers use it, 75% of Facebook's egress is served via QUIC [20].

It features flexible Connection-ID routing, Zero Downtime server restart option, multithreading with enhanced scalability on multi-core systems, or pluggable congestion control. mvfst complies with a somewhat outdated version of the standard (draft-29 as of 2022-04-02) [7].

On GitHub this project has the most forks and stars of the implementations examined in this paper [11].

picoquic. A minimalist implementation complying with the latest versions of QUIC with emphasis on non-HTTP transports, like DNS over QUIC. It uses picotls for the TLS 1.3 cryptography and considers only single threaded working modes. It supports most major features of QUIC and also implements not yet standardized features, like QUIC Multipath [21].

The picoquic library can be compiled for most major desktop operating systems from Linux to Windows [10].

quicly. A modular implementation intended for the QUIC support of the H2O HTTP server. It is currently somewhat outdated, supporting protocol version Draft-27 [7]. The vendor of quicly provides the picotls library which is used by other implementations for TLS 1.3 [9].

4. Bottlenecks and Solutions

QUIC has shown great benefits to its predecessors, however, a number of performance bottlenecks remain unsolved. This section analyzes the limitations of QUIC and investigates possible solutions to them.

Mobile Use

As discussed in Section 2.1, one of QUIC's biggest achievements is the 0-RTT handshake. However, this technological advancement does not always achieve a visible performance improvement. Mobile users often do not see the benefit of QUIC, as apps such as YouTube perform handshakes in the background. Users often take their time browsing and looking for media content, which enables a proactive connection establishment. However, this optimization is only available when the server is known beforehand. Furthermore, "mobile phones are also more CPU-constrained than desktop devices" [1], therefore benefit less when high bandwidth connection is available [1].

doi: 10.2313/NET-2022-11-1_01

Packet reordering

The packet reordering engine has great influence on the number of lost packets and the throughput of the QUIC protocol. Tests executed in both single- and multi-connection scenarios in [22] have shown that some implementations experience deduction in throughput when packets exchanged between client and server are re-ordered.

To achieve the desired throughput and congestion window size, the packet reordering engine has to be able to process out-of-order packets as timely as possible. Otherwise, packets will be regarded as lost and a new request must be filed for the lost packets. Picoquic offers insight into two different implementations of a reordering algorithm: the newer version utilizes a splay tree (self-balancing binary search tree) and an older one uses linear search. Comparing these two approaches, reveals that linear search worsens the throughput, unless there is no packet reordering required. In this case splay trees perform worse, because the whole tree has to be traversed in order to find the packet [22].

Kernel and user space bottleneck

QUIC implementations are executed in user space, to ease development and enable fast version changes, therefore, overshadowing an optimal and efficient CPU usage. Measurements show in [22] that the biggest CPU overhead occurs when the user-space QUIC stack emits the properly formatted and encrypted packets and passes them to the kernel for UDP transmission. Thus, sending a given blob via QUIC may result in an increased number of kernel system calls and the necessary data copying, causing higher CPU load.

Measurements depicted in in [1] highlight that QUIC's CPU costs were about 3.5 times bigger than TLS/TCP. However, implementations like Quant were able to partly eliminate this issue with the use of kernel-bypass techniques [22]. QUANT may be configured to use the "netmap" package which can bypass the data copy from user-space to the kernel and allows sending a large number of packets with a single call [19].

Crypto

Assessments in [22] showed that CPU utilization was at its peak when crypto methods like aead_enc() and aead_dec() were executed [22]. The aforementioned crypto functions are stateless, thus they are good candidates for offloading them to an external hardware device, e.g. to SmartNIC.

SmartNICs (Smart Network Interface Cards) are getting commodity devices in today's data centers. These hardware devices feature programmable components like an application-specific integrated circuit (ASIC), a field-programmable gate array (FPGA) or a built-in system-on-chip (SoC). These programmable features make it possible to offload some processing to the network hardware.

A possible architecture integration of QUIC with NIC is introduced in [22]. New connections are managed in a connection table and NIC is notified about every new entry. However, storing every connection is memory costly, therefore, short connections are taken care of by the host CPU. Nevertheless, the limited amount of memory of NICs still has to be taken into consideration. A suitable hash function could help to enhance memeory access.

However, there is a possibility that the NIC throughput performs worse than desired, as hardware execution time is almost ten times slower than CPUs. To resolve this throughput issue, parallelization of modules can be introduced. Further optimization enables a faster handling of incoming packets, by arranging them to the first available decoder/encoder [22].

QUIC on High bandwidth, Low-Delay, Low-Loss Networks

If employed "on networks with plentiful bandwidth, low delay, and low loss rate" in [22] QUIC does not result in performance gain. Given that a connection solely requires a few milliseconds of RTT, QUIC's 0-RTT optimization will not provide a measurable advance. In these situations, TCP may outperform QUIC, as TCP is implemented in kernel and thus provides lower CPU costs. Similarly, extremely high-bandwidth results in a big number of packets arriving in a short amount of time, causing high CPU demands. Therefore, TCP's CPU advantage could once more lead to greater performance [22].

UDP Blocking

According to the large-scale experimentation framework measurements in [1] ca. 4% of the clients were not able to communicate via QUIC. Examining the individual cases revealed that corporate networks often block or throttle UDP traffic between the Internet and the enterprise network due to security or performance reasons [1].

Some network operators may throttle UDP traffic based on security concerns and this could hinder QUIC performance or disable it [15].

It is a crucial task for the future to implement solutions for preventing UDP based attacks and yet allowing QUIC traffic. Until then, servers must support traditional TCP transport and clients should be able to fall back to TCP-based communication methods.

5. Conclusion

This paper looked at the evolution of the web protocols which led to the development of QUIC, examined the core functionality of the new protocol and inspected a handful of open source implementations.

QUIC is a promising solution for today's internet challenges. It offers new approaches to existing problems: it combats TCP's head-of-line blocking, ideally reduces round trip time, offers greater stability and better performance in many scenarios.

However, the novelty of using UDP as a transmission protocol may uncover some new issues which require attention by implementors of QUIC software and by network operators. QUIC offers enhanced congestion control and loss detection ability.

doi: 10.2313/NET-2022-11-1_01

Furthermore, the QUIC stack lives in user-space, which provides many benefits but results in extra CPU costs, causing a potential performance bottleneck. Some implementations offer options for bypassing the traditional interface between the kernel and the user-space for emitting UDP datagrams. Efforts are in progress to offload some of the CPU intensive processing of the QUIC stack to the hardware: the FPGA circuits of SmartNICs may take over some parts of 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," *Proceedings of SIGCOMM 2017, Los Angeles, CA, USA, August 21-25*, 2017.

[2] W3Techs, "Usage statistics of HTTP/3 for websites," *https://w3techs.com/technologies/details/ce-http3 as of 2022-04-02*, 2022.

[3] I. Alexa Internet, "The top 500 sites on the web," *https://www.alexa.com/topsites as of 2022-04-02*, 2022.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," *https://www.ietf.org/rfc/rfc2616.txt as of 2022-04-02*, 1999.

[5] I. H. W. Group, "This is the home page for HTTP/2, a major revision of the Web's protocol." *https://http2.github.io/ as of 2022-04-02*, 2022.

[6] M. Thomson and S. Turner, "Using TLS to Secure QUIC," *RFC 9001*, 2021.

[7] quicwg, "Implementations," *https://github.com/quicwg/base-drafts/wiki/Implementations as of 2022-04-02*, 2022.

[8] NTAP, "QUIC implementation for POSIX and IoT platforms," *https://github.com/NTAP/quant as of 2022-04-02*, 2022.

[9] T. H. project, "A modular QUIC stack designed primarily for H2O," *https://github.com/h2o/quicly as of 2022-04-02*, 2022.

[10] P. Octopus, "Minimal implementation of the QUIC protocol," *https://github.com/private-octopus/picoquic as of 2022-04-02*, 2022.

[11] F. Incubator, "An implementation of the QUIC transport protocol," *https://github.com/facebookincubator/mvfst as of 2022-04-02*, 2022.

[12] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," *RFC 9000*, 2021.

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

[14] G. Carlucci, L. D. Cicco, and S. Mascolo, "HTTP over UDP: an Experimental Investigation of QUIC," *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.

[15] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is QUIC?" *Communication QoS, Reliability and Modeling Symposium*, 2016.

[16] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3) draft-ietf-quic-http-34," *https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34 as of 2022-04-02*, 2021.

[17] Apple, "TN3102: HTTP/3 in your app," *https://developer.apple.com/documentation/technotes/tn3102-http3-in-your-app as of 2022-04-02*, 2022.

[18] T. H. project, "TLS 1.3 implementation in C (master supports RFC8446 as well as draft-26, -27, -28)," *https://github.com/h2o/picotls as of 2022-05-16*, 2022.

[19] L. Rizzo, "netmap - the fast packet I/O framework," *http://info.iet.unipi.it/ luigi/netmap/ as of 2022-04-02*, 2022.

[20] M. Joras and Y. Chi, "How Facebook is bringing QUIC to billions," *https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/ as of 2022-04-02*, 2022.

[21] Y. Liu, Y. Ma, Q. D. Coninck, O. Bonaventure, C. Huitema, and M. Kuehlewind, "Multipath Extension for QUIC, draft-ietf-quic-multipath-01," *https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/ as of 2022-04-02*, 2022.

[22] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC Quicker With NIC Offload," *Workshop on Evolution, Performance, and Interoperability of QUIC (EPIQ2020), August 10-14*, 2020.

Recycle, Reduce, Reuse - Surveying Instruction Set Architectures

Philipp Erhardt, Henning Stubbe*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: philipp.erhardt@tum.de, stubbe@net.in.tum.de

***Abstract*—Code size can play an important role when implementing emulators for an instruction set architecture (ISA), as performance can in part depend on whether cache misses occur frequently. Additionally, implementation complexity is heavily correlated with the complexity of the ISA to emulate. In this paper we take a look at different ISAs and compare their complexity as well as the object code size of an example function. A git repository is provided, allowing reproduction and adaptation for analysis of other functions. While disassembling executables, we also found a possible size optimization in a *RISC-V* compiler.**

***Index Terms*—instruction set architectures, code size**

1. Introduction

When creating emulation software, the complexity of the implementation is tightly coupled with that of the emulated ISA. High complexity can increase effort needed for testing and verification. Furthermore, reducing the size of the input machine code can improve performance due to fewer cache misses during interpretation as well as reducing memory, storage and transmission overhead.

Both properties show a tradeoff when designing ISAs: one can either use simple and usually small instructions of which more will be needed to express complex programs, or add more instructions that express the program at a more direct level. Nowadays, there are many ISAs that have taken different design decisions. Some have become symbols for high complexity due to thousands of supported instructions, others aim to provide only a minimal set. Variants for microcontrollers have been introduced to minimize code size, improving usage of limited storage.

This paper aims to describe and compare different ISAs to evaluate their fitness for the implementation of an emulator, with focus being on code size reduction and decreased implementation complexity. Section 2 explains concepts of ISAs and introduces selected ones. Section 3 discusses related work on ISA complexity and usage, Section 4 compares ISAs with regard to their complexity and size.

2. Background

The first part of this section will present common properties used to differentiate ISAs. The second part presents selected ISAs that are compared in Section 4.

2.1. Instruction Set Architecture

The ISA of a processor defines operations for loading, storing and manipulation of data, as well as how these instructions are encoded and executed [1, 5.3 Instruction Set Architecture]. ISAs can be compared by means of their properties, of which selected ones will be presented.

Internal Storage Type The differentiation between stack, accumulator or register machines is fundamental when analyzing ISAs.

In a stack machine, an instruction operates by taking zero or more arguments from the stack and pushing its result to the top of the stack. For most instructions, operands are implicit and need not be specified.

In an accumulator architecture an instruction operates on the accumulator and zero or more explicit arguments, with results being stored in the accumulator.

Instructions of register machines explicitly specify both source and destination operands. In a *register-register machine*, all operands must be registers, except for operands of load and store instructions. These architectures are called *load-store* architectures. In a *register-memory* architecture, most instructions can access memory operands [2, A.2 Classifying Instruction Set Architectures].

Instruction Encoding The instruction encoding of an architecture defines the executable format. The exact binary values of instructions differ widely between architectures. We can however differentiate between fixed-length and variable-length encodings [2, A.7 Encoding an Instruction Set].

Complexity ISAs can be classified by the complexity of their operations. The most common categories are *Reduced Instruction Set Computer* (*RISC*) and *Complex Instruction Set Computer* (*CISC*).

CISC processors are typically able to perform hundreds of instructions of differing complexity with memory operands and many addressing modes available for most. These instructions can take many clock cycles to execute.

In comparison, *RISC* processors include mostly basic instructions, more complex operations must be expressed using them. Many *RISC* architectures aim to execute one instruction per clock cycle [1, p. 91]. Often memory access is only possible via load and store instructions [3, Chapter 3].

Endianness The endianness of a processor defines the byte order when accessing data from memory. With little endian, the most significant byte is stored at the highest address. With big endian byte order, the most significant byte is stored at the lowest address [4, Section 2.4].

doi: 10.2313/NET-2022-11-1_02

Addressing Modes ISAs typically include different ways of accessing operands, either as immediate values, in registers or in memory. Memory addressing modes can express the address as *displacement* relative to a known location like the stack, as *register indirect* with an address already being stored in a register or *indexed* addressing where an offset is added to a base address (for accessing data in arrays), as well as many more addressing modes not mentioned here [2, Section A.3].

2.2. Selected ISAs

The following section briefly describes the ISAs that will be compared in Section 4. These architectures were selected for comparison because they either claim reduced object code size, have a small number of instructions or feature a design that is vastly different from the others.

ARM T32 The *ARM T32* instruction set, previously called *Thumb-2*, is a superset and successor of the *ARM Thumb* instruction set [5] featuring 301 instructions [6, p. 5-13]. Encoded instructions have variable length of either 16 or 32 bits [7]. It is in the family of *RISC* architectures. Since memory access is only possible using load and store instructions, *T32* can be classified as a load-store architecture [8, p. 36]. Registers are 32 bits in size [8, p. 38]. Processors with this architecture can switch between big-endian and little-endian mode for memory access using the `SETEND` instruction [8, p. 7569, F5.1.182].

RISC-V Compressed *RISC-V* is a free and open family of *RISC* ISAs. While any *RISC-V* ISA includes the base integer ISA, it is possible to add optional extensions [9, Section 1.3]. The size of encoded instructions is fixed to 32 bits for the base ISA, extensions can however use a multiple of 16 bits to encode further instructions [9, Section 1.5]. One extension providing 16-bit encodings for common instructions is the standard "C" ("Compressed") extension [9, Chapter 16]. The *RV32I Base Integer Instruction Set* defines 40 instructions [9, p. 31] and is little endian [9, p. 8]. *RV32I* is load-store as memory access is only possible using load/store instructions. All other instructions use registers as operands, so these ISAs can be classified as register machines [9, p. 42].

WebAssembly WebAssembly is an open standard for a "virtual instruction set architecture" based on a stack machine [10, p. 5-7]. Instruction opcodes are one or two bytes long, but since immediate arguments can follow these opcodes, the instruction set is variable-length [10, Section 5.4]. Memory access is possible via load-store instructions, the byte order is little endian [10, p. 22]. There are 437 valid opcodes, with an additional 69 being currently reserved [11].

x86-64 The *x86-64* instruction set is a backwards-compatible successor of the *Intel 8086*'s architecture [12, p. 37]. Instruction size is variable and ranges from 1 to 15 bytes [12, p. 3058], the byte order is little endian [12, p. 32]. The basic ISA provides 16 general purpose registers, each having a width of 64 bits [12, p. 76]. The internal storage type for *x86* is a register-memory machine [2, Figure A.3], with some instructions like `MUL` and `DIV` using implicit accumulator registers as operands and destinations [12]. Heule et al. count at least 981 mnemonics and 3,684 instruction variants [13], allowing classification as *CISC*.

Z80 The Zilog *Z80* is a microprocessor introduced in 1976, with later versions still being used today [14]. It features 158 instructions of which 78 are adapted from the Intel 8080 CPU, making the instruction set backwards-compatible [15, p. 46]. Instruction size is variable ranging from 1 to 4 bytes [15, p. 57]. They can operate on register or memory operands, with some storing results in accumulator registers [15, p. 40-47]. Featuring instructions like `LDIR` that can occupy the CPU for many clock cycles [15, p. 41], this instruction set can be classified as *CISC*.

3. Related Work

In [16], Davidson and Vaughan analyze the relation between instruction set complexity and program size. A technique called "instruction set subsetting" is used to eliminate biases that could arise when comparing different architectures. Three subsets of the rather complex *VAX* instruction set are created with decreasing complexity: while *MAXVAX* supports 16 addressing modes both in source and destination operands for almost all instructions, *MIDVAX* supports only eight addressing modes and restricts destination operands to registers only. Some more complex instructions are not available at all. The *MINVAX* instruction set further reduces available instructions and addressing modes, memory access is only possible via load-store instructions. Their comparison of the object code size of ten different programs shows an increase in average code size with reduced architectural complexity: compared to the baseline *MAXVAX* instruction set, programs compiled for *MIDVAX* are on average 1.54 times the size, while the average size of programs compiled for *MINVAX* grows to 2.48 times. They also note that average instruction sizes are 4.10 bytes for *MAXVAX*, 3.71 bytes for *MIDVAX* and 3.61 bytes for *MINVAX*, showing that the compiler is able to use more complex and large instructions when they are available.

To reduce implementation complexity, only a subset of an instruction set could be implemented. In [17], Akshintala et al. analyze the distribution of instruction opcodes in Linux packages for the *x86-64* architecture. A table [17, Table 5] shows the number of instructions needed to support a given percentage of available packages. They find that an emulator aiming to run 80% of available packages would have to implement 189 instruction mnemonics, while for 90% of packages 230 mnemonics are required. 611 additional instructions are needed for full compatibility. They also recommend a sequence of instructions that can be used for such an implementation based on the popularity of packages using these instructions.

Another approach for reducing complexity or effort when implementing an emulator is using a very small ISA, like *LC-3* described by Yale N. Patt and Sanjay J. Patel in [18, pp. 520-545]. With a total of 16 opcodes, one of them reserved for the future, the instruction set is very small. It features eight 16-bit general purpose registers and is a load-store architecture [18, p. 553]. All instructions are 16 bits wide, with the upper 4 bits defining the opcode. Originally it was planned to include this architecture in the comparison in Section 4. However, due to a lack of functioning compilers from *C* to *LC-3*, it did not end up being included.

doi: 10.2313/NET-2022-11-1_02

4. Comparison

This section compares the different ISAs introduced in Section 2.2 with regard to their complexity and code size.

4.1. Complexity

In order to assess the complexity required for an emulator implementation, we can look at the complexity of the target ISA. The distinction between *CISC* and *RISC* is on a high level, but can be helpful for estimating the complexity of operations before implementing them.

A typical difference is the concept of a load-store architecture (usually used in *RISC* ISAs) versus memory operands. In *CISC* architectures, it is often possible to directly operate on memory operands, requiring only one instruction for manipulation. Load-store architectures however must explicitly load and store data for manipulation in a register. Even if the encoding for the *CISC* instruction were larger in size than a typical *RISC* instruction, encoding only a single instruction instead of three can still result in smaller code. In fact, as shown in Figure 1, incrementing a value at a given memory address stored in a register (`r0` for *ARM T32*, `rdi` for *x86-64*) needs triple the amount of bytes to encode for the *RISC* architecture in this case.

03 68 `ldr r3, [r0, #0]`	**ff 07** `incl (%rdi)`
01 33 `adds r3, #1`	
03 60 `str r3, [r0, #0]`	

Figure 1: Comparing *ARM T32* (left) and *x86-64* (right)[1]

Another method for comparing the complexity of different ISAs is comparing the number of operations that are possible, including addressing modes. However, getting an accurate and up to date count of available opcodes for different architecture variants is challenging: as Heule et al. note in [13], as well as Mahoney and McDonald in [19], getting accurate data on *x86-64* opcodes is hard. Similar problems with other architectures, especially the inaccessibility of PDF files for programmatically counting opcodes, have prevented further analysis in this paper.

4.2. Code Size

To compare code size we compiled a simple *C* function for different architectures. The code is shown in Figure 2. A git repository containing all code, including Makefiles for generating the results for all architectures, is available online at https://github.com/xarantolus/iitm-surveying-isas.

Tools This section will outline the tools used for compiling and measuring program size for each architecture. The installation steps are also available in the git repository linked above.

All compilations were done on an *Ubuntu 20.04.4 LTS x86_64* system. Where available, the `-Oz` option is passed to compilers to "optimize aggressively for size rather than speed", else the `-Os` flag is used to "optimize for size" [20]. The following lists the tools used for compiling the C program for each architecture. More detailed installation instructions can be found within the git repository.

```
int fib(int n) {
    if (n <= 1) { return n; }
    int prev = 0; int current = 1; int tmp;
    for (int i = 2; i <= n; i++) {
        tmp = current;
        current += prev;
        prev = tmp;
    }
    return current;
}
```

Figure 2: Fibonacci function written in C

ARM T32 The compiler used is *arm-linux-gnueabihf-gcc 9.4.0* from the *gcc-arm-linux-gnueabihf* package.

RISC-V Compressed The *riscv64-unknown-linux-gnu-gcc (g5964b5cd727) 11.1.0* compiler from the *RISC-V* toolchain available at [21] was used to compile the program. An install script is available in the `riscv` directory of the git repository.

WebAssembly Instructions for installing the *emcc 3.1.8* compiler are available at [22].

x86-64 For compilation *gcc 9.4.0* from the *gcc* package is used.

Z80 The *clang 12.0.0* compiler is built from the project at [23]. Installation steps were adapted from [24].

Results Table 1 shows the object code size of the compiled `fib` function from Figure 2.

The *ARM T32* object code includes 14 instructions with a size of 2 bytes each, making it the smallest result. No 4-byte instructions were needed, showing that the compiler was able to take advantage of the 16-bit instruction variants.

Similarly, the compiled output for *RISC-V Compressed* uses mostly 16 bit instructions, except for two 4-byte `BGE` instructions used to implement branches. One interesting detail is that the compiler generates two `RET` instructions at the end of the function, adding an extra two bytes while only one instruction would have sufficed. A maintainer of the *RISC-V* toolchain responded to our inquiry, calling this behavior "a missed optimization opportunity" [25].

The *WebAssembly* version uses mostly two-byte instructions. Despite the smaller average instruction size, its comparatively high number of instructions results in the largest output size.

The output compiled for *x86-64* contains 14 instructions with a length of 1 to 5 bytes. Nine instructions have a length of 2 bytes. Two `MOV` instructions are 5 bytes long as they encode a 4-byte immediate value. Additionaly, there is one single-byte, one three-byte and one four-byte instruction present. The small instruction count compared with *WebAssembly* and *Z80* leads to the overall smaller code size, despite the larger average instruction size.

While the instructions for the *Z80* version have an average size of only 1.88 bytes, it needs 34 instructions to implement the function, resulting in the second largest code size.

1. Adapted from [2, Figure A.2]

doi: 10.2313/NET-2022-11-1_02

TABLE 1: Object code size of the fibonacci function from Figure 2 compiled for different architectures

Architecture	Compiler	Size	Instruction Count	Avg. Instruction Size
ARM T32	*arm-linux-gnueabihf-gcc 9.4.0*	28B	14	2.00B
RISC-V Compressed	*riscv64-unknown-linux-gnu-gcc 11.1.0*	30B	13	2.31B
WebAssembly	*emcc 3.1.8*	71B	37	1.92B
x86-64	*gcc 9.4.0*	36B	14	2.57B
Z80	*clang 12.0.0*	64B	34	1.88B
ARM A32	*arm-linux-gnueabihf-gcc 9.4.0*	52B	13	4.00B
RISC-V	*riscv64-unknown-linux-gnu-gcc 11.1.0*	52B	13	4.00B

In addition to the ISAs presented in Section 2.2, Table 1 also shows the code size of *ARM A32* and *RISC-V* without the "C" extension. Both ISAs have 32-bit wide fixed-length instructions [9, p. 25] [26].

ARMs' claim of code size reduction of the variable-length *ARM T32* ISA compared to *ARM A32* [7] holds true in this case with a reduction by almost half.

The code size for *RISC-V Compressed* is approximately 40% smaller than the size for its fixed-length *RISC-V* counterpart, exceeding the "25%-30% code-size reduction" claim for the compressed extension in this case [9, p. 115].

Limitations This comparison is rather limited as only one small program is compared using only one compiler for each architecture. In addition, different compilers might support different optimizations regarding the program size.

Comparison with Previous Work As mentioned in Section 3, Jack W. Davidson and Richard A. Vaughan found in [16] that reduced architectural complexity leads to larger overall code size.

If we classify *WebAssembly* as closer to *RISC* than *CISC* due to it being a load-store architecture, it fits this observation. However, the *Z80* instruction set also produces an overall large code size despite it being classified as *CISC*. Additionally, contrary to the expectations one might have on the basis of the mentioned study, the *RISC* architectures *ARM T32* and *RISC-V Compressed* result in the overall smallest code size.

One explanation for these findings is the limited example function. Its small amount of required variables makes it possible for all data to be stored in registers, foregoing the need to access memory. As mentioned in Section 4.1, requiring more instructions to read and write memory in *RISC* ISAs could lead to larger code size. Since no explicit memory accesses are necessary for the `fib` function, the disadvantages of *RISC* ISAs do not come into consideration in this case.

5. Conclusion and Future Work

In this paper we analyzed the object code size of a program when compiled for different architectures, creating a system of Makefiles that can be adapted for analyzing other *C* functions as well. While doing so, we found variations in the code size when compiling for different architectures and a possible size optimization in a *RISC-V* compiler.

Our example code reached the smallest sizes when compiled for the *ARM T32* and *RISC-V Compressed* architectures, indicating that they can be suitable for reducing overhead and cache misses in an emulator implementation.

However, the selection of ISAs is not the only factor when reducing code size. Future work can explore reducing complexity by limiting the set of opcodes and addressing modes available to a compiler when translating a program. The effect of register spilling on code size, especially for load-store architectures that need more instructions to operate on memory operands, can also be explored. Additionally, analyses similar to [17] by Akshintala et al. can be done for other ISAs before implementing an emulator.

References

[1] C. Douglas, *Essentials of Computer Architecture.* Chapman and Hall/CRC, 2017, vol. Second edition.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A quantitative approach*, 5th ed. Morgan Kaufmann, 2011.

[3] J. Ledin, *Modern Computer Architecture and Organization*. Packt Publishing, 2020.

[4] J. Catsoulis, *Designing embedded hardware*, 2nd ed. O'Reilly, 2005.

[5] Arm Limited, "About Thumb-2," https://developer.arm.com/documentation/ddi0308/d/Introduction-to-Thumb-2/About-Thumb-2?lang=en, [Online; accessed 31-March-2022].

[6] ——, *Arm® A32/T32 Instruction Set Architecture*, December 2021.

[7] ——, "T32 Instruction Set," https://developer.arm.com/architectures/instruction-sets/base-isas/t32, [Online; accessed 24-March-2022].

[8] ——, *Arm® Architecture Reference Manual for A-profile architecture*, 2022.

[9] Editors A. Waterman and K. Asanović, RISC-V Foundation, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2019121*, December 2019.

[10] WebAssembly Community Group, *WebAssembly Specification, Release 1.1 (Draft 2022-03-21)*, https://webassembly.github.io/spec/core/_download/WebAssembly.pdf, March 2022, [Online; accessed 1-April-2022].

[11] ——, "Index of Instructions – WebAssembly 1.1 (Draft 2022-03-21)," https://webassembly.github.io/spec/core/appendix/index-instructions.html, [Online; accessed 1-April-2022].

[12] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, 2021.

[13] S. Heule, E. Schkufza, R. Sharma, and A. Aiken, "Stratified synthesis: Automatically learning the x86-64 instruction set," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 237–250. [Online]. Available: https://doi.org/10.1145/2908080.2908121

[14] D. M. G. Preethichandra, "Z80—the 1970s microprocessor still alive," *IEEE Micro*, vol. 41, no. 6, pp. 156–157, 2021.

[15] Z80 Microprocessors, *Z80 CPU User Manual UM008011-0816*, http://www.zilog.com/docs/z80/UM0080.pdf, [Online; accessed 30-March-2022].

[16] J. W. Davidson and R. A. Vaughan, "The effect of instruction set complexity on program size and memory performance," *ACM SIGOPS Operating Systems Review*, vol. 21, no. 4, pp. 60–64, Oct. 1987. [Online]. Available: https://doi.org/10.1145/36204.36184

[17] A. Akshintala, B. Jain, C.-C. Tsai, M. Ferdman, and D. E. Porter, "X86-64 instruction usage among c/c++ applications," in *Proceedings of the 12th ACM International Conference on Systems and Storage*, ser. SYSTOR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 68–79. [Online]. Available: https://doi.org/10.1145/3319647.3325833

[18] Y. Patt and S. Patel, *Introduction to Computing Systems: From Bits and Gates to C and Beyond*, 2nd ed. McGraw-Hill Higher Education, 200.

[19] W. Mahoney and J. T. McDonald, "Enumerating x86-64–it's not as easy as counting."

[20] GCC team, "Optimize Options (Using the GNU Compiler Collection (GCC))," https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html, [Online; accessed 4-April-2022].

[21] RISC-V Software Collaboration, "GNU toolchain for RISC-V, including GCC," https://github.com/riscv/riscv-gnu-toolchain, [Online; accessed 8-April-2022].

[22] The Emscripten project, "Emscripten SDK," https://github.com/emscripten-core/emsdk, [Online; accessed 8-April-2022].

[23] Retro Computing at Georgia Tech, "The LLVM Compiler Infrastructure," https://github.com/gt-retro-computing/llvm-project, [Online; accessed 8-April-2022].

[24] The IMSAI Gang, "LLVM Z80: Building," https://imsai.dev/posts/build_llvm/, [Online; accessed 8-April-2022].

[25] "Duplicate ret instruction at end of function," https://github.com/riscv-collab/riscv-gnu-toolchain/issues/1048, [Online; accessed 8-April-2022].

[26] Arm Limited, "A32 Instruction Set," https://developer.arm.com/architectures/instruction-sets/base-isas/a32, [Online; accessed 5-April-2022].

Network Path Monitoring

Buse Barcin Halis, Florian Wiedner*, Max Helm*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge23jod@mytum.de, wiedner@net.in.tum.de, helm@net.in.tum.de

***Abstract*—Network path monitoring is an important feature of modern networks. It enables to understand the behavior of the network. However, a network is a complex structure, and therefore it is a challenge to measure network characteristics from only the endpoints.**

Since the early days of the Internet, various network monitoring methods have been proposed. This paper focuses on methods for measuring network metrics such as packet loss, packet reordering, point of failure, round trip time, and bottleneck router buffer size. The differences in their implementation are highlighted, and some of their limitations are pointed out. It is concluded that OneProbe is a reliable method, but needs further development to measure more metrics.

***Index Terms*—network path monitoring, network metrics, network measuring methods**

1. Introduction

Networks are the foundation of many applications. When a problem occurs in the network, the state of the network directly affects the applications. Therefore, it is important to monitor networks and gain insights into the network state. This makes it possible to understand the behavior of the network, measure network performance, identify the problems of a network and find the causes for the problems. However, monitoring the network path is a difficult process. The network is complex, and when a fault occurs, it is difficult to determine what the problem is or where in the network the problem occurs. Therefore, we need accurate and efficient methods to monitor network paths.

Many methods have been proposed that focus on measuring network metrics [1]–[4]. This paper presents network monitoring methods that focus on the metrics of packet loss, packet reordering, point of failure, round trip time, and bottleneck router buffer size.

The rest of this paper is structured as follows: Section 2 provides background information. In Section 3, the detailed process of the methods for each presented metric is explained. Section 4 provides a comparison between the introduced methods, and Section 5 concludes the paper.

2. Background Information

In this section, network metrics are defined for which measurement methods are proposed in the next section.

Packet Loss: Loss of data packets during transmission on the network path from the source to the destination and non-arrival of the data packet at the destination [5].

Packet Reordering: The arrival of data packets at the destination not in the same order as they were sent from the source [5].

Failure Point: The link where a network problem occurs, such as packet loss or reordering [2].

Round Trip Time (RTT): Time interval between sending a packet from the source to the destination and receiving an acknowledgement for this packet at the source from the destination [1].

Buffer Size: The buffer is a storage area on a router where the data packets are temporarily stored. The buffer size indicates the capacity of this storage area [4].

3. Network Path Monitoring Methods

This section presents the methods categorized according to their measured metrics.

3.1. Packet Loss

Many methods have been developed to measure packet loss behavior in network paths. In this subsection, three of them are introduced, namely OneProbe, Tulip and Sting.

OneProbe: OneProbe is a TCP probing method for monitoring network paths and measuring network metrics, proposed by Luo et al. [1]. OneProbe uses TCP data probes. Each probe sent to the destination contains two TCP data packets as probe packets and triggers two new TCP data packets from the destination as response packets. The probe packets are used to inspect the forward path, while the response packets are used to inspect the reverse path. The probing process works as shown in Figure 1. The TCP data probe packets sent by OneProbe are declared as Cm|n and the TCP data response packets sent by the server are declared as Sm|n, where m corresponds to the sequence number and n to the acknowledgment number of a TCP data segment. Ŝm|n denotes a data retransmission.

There are 5 different possibilities for two packets on a path:

1) F0/R0: The server/OneProbe receives both probe/response packets in correct order.
2) FR/RR: The server/OneProbe receives both probe/response packets in reverse order.
3) F1/R1: The server/OneProbe receives only the second probe/response packet, the first packet is lost.

doi: 10.2313/NET-2022-11-1_03

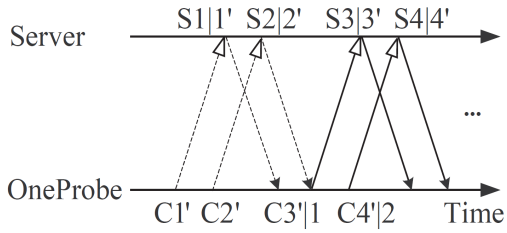


Figure 1: Probing process of OneProbe [1]

TABLE 1: 18 path events in one round of OneProbe [1]

Path events	1st response packets	2nd response packets	3rd response packets
1. F0 x R0	S3|3'	S4|4'	-
2. F0 x RR	S4|4'	S3|3'	-
3. F0 x R1	S4|4'	Ŝ3|4'	-
4. F0 x R2	S3|3'	Ŝ3|4'	-
5. F0 x R3	Ŝ3|4'	-	-
6. FR x R0	S3|2'	S4|2'	Ŝ3|4'
7. FR x RR	S4|2'	S3|2'	Ŝ3|4'
8. FR x R1	S4|2'	Ŝ3|4'	-
9. FR x R2	S3|2'	Ŝ3|4'	-
10. FR x R3	Ŝ3|4'	-	-
11. F1 x R0	S3|2'	S4|2'	Ŝ3|2'
12. F1 x RR	S4|2'	S3|2'	Ŝ3|2'
13. F1 x R1	S4|2'	Ŝ3|2'	-
14. F1 x R2	S3|2'	Ŝ3|2'	-
15. F1 x R3	Ŝ3|2'	-	-
16. F2 x R0	S3|3'	Ŝ2|3'	-
17. F2 x R1	Ŝ2|3'	-	-
18. F3	Ŝ1|2'	-	-

4) F2/R2: The server/OneProbe receives only the first probe/response packet, the second packet is lost.
5) F3/R3: The server/OneProbe receives no probe/response packet, both packets are lost.

There are 18 possible scenarios for packet loss and reordering in a probe round, resulting from the combination of the above events on the forward and reverse paths. These combinations and the response packets that the source host receives for all these 18 scenarios are listed in Table 1. Based on the response packets the source receives, OneProbe can identify which scenario is present and whether there is packet loss or reordering in the forward or reverse path. Scenarios 11 to 18 describe the scenarios of packet loss in the forward path. Scenarios 3, 4, 5, 8, 9, 10, 13, 14, 15, 17 describe the scenarios of packet loss in the reverse path. There are only 3 cases where the scenario cannot be distinguished because the responses are not unique:

- Scenario 14 and 15
- Scenario 12 and 13
- Scenario 5 and 10

The packet loss rate is measured by OneProbe by sending successive probe rounds to the destination. OneProbe only considers the first packet for measuring the loss rate. The packet loss rate for the forward path is calculated as follows:

$$\frac{\text{\#Probe rounds with first probe packet loss}}{\text{\#Total probe rounds}} \quad (1)$$

The packet loss rate for the reverse path is calculated as follows:

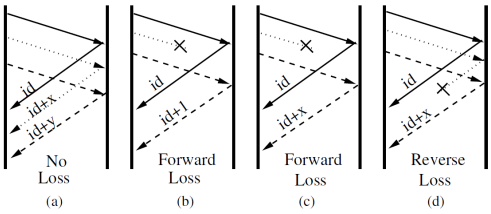


Figure 2: Packet loss scenarios in Tulip [2]

$$\frac{\text{\#Probe rounds with first response packet loss}}{\text{\#Total probe rounds}} \quad (2)$$

Tulip: One of the other tool which detects packet loss on a network path is Tulip proposed by Mahajan et al. [2]. Except detecting packet loss, Tulip can also identify the location of the packet loss within three hops. Tulip measures network characteristics in the forward path. If we want to meaure the reverse path, tulip can be used at the destination point. Tulip uses the IP identifier counters feature of routers to detect packet loss. Each IP packet contains a unique identification field (IP-ID) to enable IP fragments to be reassembled. Most routers implement the IP-ID using a counter, and the IP-ID is incremented with each packet generated. Tulip exploits these IP-IDs.

The loss detection mechanism of Tulip works as follows: The source sends three probe packets: two control packets and one data packet in the middle, to the router. Different protocols such as UDP, TCP or ICMP can be used for these probe packets. Each of these probe packets generates a response packet from the router. There are 3 possibilities for the data packet loss, as shown in Figure 2:

1) The source receives all three responses: No Loss
2) The source receives only two responses triggered by control packets and the IP-IDs of the response packets are consecutive: Forward Loss
3) The source receives only two responses triggered by control packets and the IP-IDs of the response packets are not consecutive: Indistinguishable whether forward loss or reverse loss

A few prerequisites exist for Tulip's loss detection mechanism: The control packets should always be retained, because if a control packet or its response is lost, it is not possible to detect the direction of data loss. The probe packets should arrive at the router close together in time and in the correct order so that they can obtain consecutive IP-IDs. The packet loss rate is calculated as follows:

$$\frac{\text{\#Probe rounds with forward loss}}{\text{\#Total probe rounds}} \quad (3)$$

Sting: Sting, introduced by Savage, is another tool that can measure packet loss rates along both the forward and reverse paths between a source and a destination [3]. Sting's loss deduction algorithm measures packet loss rate by leveraging the features of the TCP protocol. For packet loss measurement, Sting uses TCP acknowledgments.

The algorithm measuring the loss rate on forward path consists two phases, namely *data seeding* and *hole filling*. In the data seeding phase, sequential TCP packets are

doi: 10.2313/NET-2022-11-1_03

sent from the source to the destination. In the hole filling phase, the source sends another TCP data packet with a sequence number one higher than the last TCP data packet in the data seeding phase. If the source receives an acknowledgment for this packet, it concludes that no packet was lost in the data seeding phase. If the source receives a duplicate acknowledgement, it means a packet loss, and the number of the acknowledgement indicates which packet was lost. The source resends the corresponding packet. This process is continued until the last data packet sent in the data seeding phase is acknowledged. In this way, the total number of lost data packets is obtained.

Measuring the loss rate in the reverse path can be problematic. The source cannot count the acknowledgments that the destination sends. This is where ack parity is used. Sting ensures ack parity using a method that will not be elaborated on here. Ack parity guarentees that destination sends an acknowledgment for every packet it receives.

Five attributes are defined for the calculation of the forward and backward path loss rate:

dataSend: The total number of data packets sent from source to the destination, can be measured directly at the source.

dataLost: The total number of lost data packets measured with Sting's loss deduction algorithm.

dataReceived: dataReceived = dataSend - dataLost

ackSent: ackSent = dataReceived. Due to ack parity, an acknowledgement is issued for each received packet.

ackReceived: The total number of acknowledgments that have reached the source, can be measured directly at the source.

The loss rate for the forward path is calculated as follows:

$$1 - (\text{dataReceived}/\text{dataSent}) \tag{4}$$

The loss rate for the reverse path is calculated as follows:

$$1 - (\text{ackReceived}/\text{ackSent}) \tag{5}$$

3.2. Packet Reordering

Many methods have been developed to measure packet reordering behavior on network paths. In this subsection, two of them are introduced, namely OneProbe and Tulip.

OneProbe: All 18 possible scenarios for packet loss and reordering in a probe round for the OneProbe method have already been shown in Table 1. Scenarios 6 to 10 describe the scenarios of packet reordering in the forward path. Scenarios 2, 7, 12 describe the scenarios of packet reordering in the reverse path. The packet reordering rate is measured by OneProbe by sending successive probe rounds to the destination. OneProbe only considers the first packet for measuring the loss rate.

The packet reordering rate for the forward path is calculated as follows:

$$\frac{\#\text{Probe rounds with reordered probe packets}}{\#\text{Total probe rounds}} \tag{6}$$

The packet reordering rate for the reverse path is calculated as follows:

$$\frac{\#\text{Probe rounds with reordered response packets}}{\#\text{Total probe rounds}} \tag{7}$$

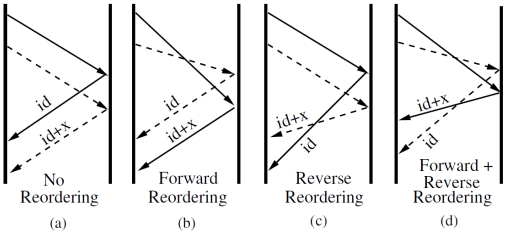


Figure 3: Packet reordering scenarios in Tulip [2]

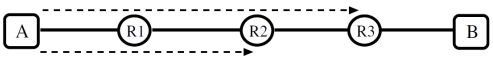


Figure 4: Building Blocks: The properties of link R2 → R3 can be estimated by subtracting the measured properties of path A → R2 from the measured properties of path A → R3. [2]

Tulip: Tulip uses IP-IDs to obtain information about packet reordering on the forward path [2]. To measure the reordering of the packets, the source sends two probe packets to the router. Each of these probe packets generates a response packet from the router that contains the headers of the probe packets so that they can be differentiated. Tulip uses IP-IDs to obtain information about the order in which packets reach the router.

There are 4 possibilities to reorder these two packages in the forward and backward paths, as shown in Figure 3:

1) No reordering: The responses are received in order and the second probe's response has a higher IP-ID.
2) Forward path reordering: The second probe's response has a lower IP-ID and reaches the source first.
3) Reverse path reordering: The second probe's response has a higher IP-ID but reaches the source first.
4) Forward and reverse path reordering: The second probe's response reaches the source second but has a lower IP-ID.

The packet reordering rate is calculated as follows:

$$\frac{\#\text{Reordered probe pairs}}{\#\text{Probe pairs for which both responses are received}} \tag{8}$$

3.3. Point of Failure for Packet Loss and Packet Reordering

The characteristics of the single links in the network can not be measured individually. In order to locate the point of a failure, we first make forward path measurements to both ends of a link and then compare the resulting measurements. This method is called Building Blocks, and is illustrated in Figure 4.

Tulip: The third metric that Tulip can measure is the point of failure on the network path using the Building Blocks method [2]. Tulip has 2 steps for locating the failure point. In the first step, the path from the source to the destination is determined using traceroute. In the second step, tulip performs either a parallel search or a binary search. In parallel search, the forward path to each router is measured one after another. In binary search, the forward path to the destination is measured first. If there is

doi: 10.2313/NET-2022-11-1_03

a fault, the forward path to middle point is measured, and depending on which part of the path contains the fault, the measurement continues with that part, and this continues recursively until the faulty link is found.

3.4. Round Trip Time

OneProbe: The third metric that OneProbe can measure is round trip time (RTT) [1]. OneProbe only considers the first packet for measuring round trip time. This is because the RTT of the second probe packet can be affected by the first packet [6]. The RTT is calculated as follows:

$$\begin{matrix}\text{First response packet}\\ \text{receive time}\end{matrix} \quad - \quad \begin{matrix}\text{First probe packet}\\ \text{sending time}\end{matrix} \tag{9}$$

3.5. Bottleneck Router Buffer Size

Loss Pairs: Liu and Crovella develop a tool called Loss Pairs for specify network characteristics such as the packet dropping behavior of a bottleneck router [4]. A loss pair defines a pair of packets where exactly one of the packet is discarded in the network while they were traveling on the same path and were close to each other in time [4]. The network conditions observed by these two packets are very similar. Thus, if one of the packets gets lost, very accurate estimates of the network conditions at the time of packet loss can be made based on the residual packet. This idea of loss pair method is used to identify router properties in the network and estimate the buffer size at a bottleneck router, assuming that most packet losses and delays occur at the bottleneck. If one packet of the pair is dropped, it indicates that the queue of the bottleneck router is full, and the calculated RTT for the residual packet includes the drain time of a full queue on the router. Therefore, the focus lies on the round trip time of the residual packet. The round trip time is obtained from the TCP data packets and their acknowledgments. And the calculated RTT is used to estimate the buffer size. It is assumed that we already know the bandwidth of the bottleneck link and the propagation delay along the path. Then the buffer size is calculated as follows:

$$\text{Bandwidth} \cdot (\text{RTT} - \text{Propagation Delay}) \tag{10}$$

4. Comparison of the Methods

Traditional ICMP-based tools such as Ping and Traceroute work universally and can be run on only one endpoint, but they provide limited and inaccurate results [3]. One problem with ICMP-based tools is that it is impossible to determine whether packet loss has occurred on the forward or reverse path [3]. The second problem is that routers and end hosts do not always respond to ICMP Ping and Traceroute [7], resulting in an inflated packet loss rate.

Sting overcomes these problems by exploiting the properties of the TCP protocol. Sting can distinguish in which direction packet loss occurs while still running on only one endpoint. One problem with Sting is that Sting uses TCP ACKs on the reverse path, even though it uses TCP data packets on the forward path. That is why Sting's reverse path measurement does not support different response packet sizes. Moreover, Sting's TCP ACKs based measurement fails for large response packet sizes. Because the TCP ACKs based measurement of reverse path loss may be underestimated for larger packets [8]. In an evaluation, Sting shows a failure rate of 54.8% for 41-byte probes and nearly 100% failure rate for 1053-byte probes [1].

Tulip can measure multiple metrics. Compared to Sting, it uses different patterns of probes. Sting targets end hosts running TCP-based servers on known ports, while Tulip can be used with both routers and hosts. Tulip provides reliable results for TCP data packets, but unreliable results for other packet types, such as UDP packets [9], since most routers in the network do not respond to UDP packets [10]. In addition, Tulip requires routers to support consecutive IP-ID measurement. Router without IP-ID counters do not support Tulip. In an evaluation, Tulip fails 80% of the time on packet loss and reordering measurements, 50% of the failures are due to Tulip using UDP probes, and 30% are due to some routers not supporting IP-ID measurements [1]. Also, Tulip cannot measure some packet loss scenarios [2].

Loss pairs enable the determination of router characteristics that were previously not directly measurable, such as the buffer size of bottleneck routers in the network. This method provides sufficiently accurate and robust results over a wide range of network configurations as well as under noisy network conditions [4]. In [4], it is claimed that whether the remaining packet is the first or the second in a loss pair makes no difference in determining the queuing delay of a congested router. However, an analysis performed in [11] shows that using the delay of the first packet tends to be more accurate than the delay of the second packet.

OneProbe is another tool capable of measuring multiple metrics, like Tulip. The packet sizes in OneProbe are configurable so that it can measure path metrics with different response packet sizes. This feature of OneProbe is used in [11] to confirm that the accuracy of delay estimation generally increases with a smaller packet size. OneProbe also overcomes some limitations of Tulip: OneProbe can measure multiple path metrics on the forward and reverse paths simultaneously with the same probe. Tulip's probe packets, on the other hand, differ in the loss and reordering measurements in the method itself. One problem with OneProbe is that it cannot distinguish some path events due to the ambiguity of some responses.

5. Conclusion and future work

In this paper an overview of network path monitoring methods with respect to packet loss, packet reordering, point of failure, round trip time, and bottleneck router buffer size is given. The technniques they use are explained, their missing features are pointed out, and a comparison between them is made.

OneProbe is the most reliable of the presented methods and provides correct and accurate results. It was tested on 39 systems and 35 web servers and found to be successful [1]. However, it does not cover a wide range of metrics. It could be promising to develop it to include more metrics. In the future, existing methods can be

further developed to overcome their limitations. Then, attempts can be made to add more metrics from other methods to OneProbe. In this way, a reliable method with a wide range of metrics can be developed.

References

[1] X. Luo, E. W. Chan, and R. K. Chang, "Design and Implementation of TCP Data Probes for Reliable and Metric-Rich Network Path Monitoring," in *USENIX Annual Technical Conference*, 2009.

[2] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "User-level Internet Path Diagnosis," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 106–119, 2003.

[3] S. Savage, "Sting: a TCP-based Network Measurement Tool," in *USENIX symposium on Internet Technologies and Systems*, vol. 2, 1999, pp. 7–7.

[4] J. Liu and M. Crovella, "Using Loss Pairs to Discover Network Properties," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001, pp. 127–138.

[5] A. Lamberti, "How to Measure Network Performance: 9 Network Metrics," https://obkio.com/blog/how-to-measure-network-performance-metrics/#how-to-measure-network-performance, 2022, [Online; accessed 01-April-2022].

[6] J.-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," in *Conference proceedings on Communications architectures, protocols and applications*, 1993, pp. 289–298.

[7] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute Probe Method and Forward IP Path Inference," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, 2008, pp. 311–324.

[8] S. Floyd and E. Kohler, "Tools for the Evaluation of Simulation and Testbed Scenarios," Internet Draft: draft-irtf-tmrg-tools, Tech. Rep., 2008.

[9] C. Parsa and J. Garcia-Luna-Aceves, "TULIP: A Link-Level Protocol for Improving TCP over Wireless Links," in *WCNC. 1999 IEEE Wireless Communications and Networking Conference (Cat. No. 99TH8466)*, vol. 3. IEEE, 1999, pp. 1253–1257.

[10] A. Haeberlen, M. Dischinger, K. P. Gummadi, and S. Saroiu, "Monarch: A Tool to Emulate Transport Protocol Flows over the Internet at Large," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 105–118.

[11] E. W. Chan, X. Luo, W. Li, W. W. Fok, and R. K. Chang, "Measurement of Loss Pairs in Network Paths," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 88–101.

Accuracy Tradeoffs of Federated Learning approaches

Ilia Khitrov, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ilia.khitrov@tum.de, christopher.harth-kitzerow@net.in.tum.de

***Abstract*—Federated Learning is a technique that implements distributed privacy preserving machine learning principles. The idea is useful for scenarios where data owners do not want to reveal private data, but a model that takes into account multiple databases is desired. However, a trade off between accuracy and privacy is implied for such approaches. We review well known algorithms of federated learning including Federated averaging, FedSGD, FedProx, FedNOVA, SCAFFOLD and SecureBoost, and compare their accuracy with each other and with a minibatch baseline. Ranging scenarios for applications of Federated Learning algorithms are discussed and existing theoretical advantages and limitations are mentioned.**

***Index Terms*—federated learning, benchmarks, accuracy**

1. Introduction

We live in a data driven world where every day enormous amounts of data are collected directly from our mobile devices and personal computers. Analyzing this data is a rapidly developing area both in research and application fields widely known as machine learning (ML). The most common techniques in ML today rely on centralized approaches, which impose problems with privacy in case of data leakage and can require powerful computation units [1]. Federated Learning (FL) is an approach that attempts to solve both of these problems by using distributed computations and encrypted information exchange between these units. In recent years, a lot of research was dedicated to developing algorithms for privacy-preserving machine learning, and FL has been proved to be a reliable and efficient way to deal with such challenges [2].

In this paper we will review the existing approaches for Federated Learning, key ideas behind them, and scenarios in which they are the most effective, as well as compare their performance with models trained on centralized data. We will start by introducing to the readers the ideas behind FL approaches and categorization of such algorithms.

2. Overview of Federated Learning

2.1. Definition of FL and notation

In a typical application scenario [1], the goal of Federated Learning algorithm is to train a model by collecting training information from distributed devices without revealing actual training datasets to the organizer. Let K be the set of data owners (clients), indexed as k, with corresponding datasets $D_1, ...D_k$; none of data owners has direct access to other clients' data. The algorithm consists of the following basic steps [1] [2]:

1) Clients for next training iteration are selected by the server;
2) Current machine learning model W is communicated to selected clients;
3) Each client k keeps their local databases D_k private and uses them to update weights of individual models W_k;
4) Server collects local models $W_1, ...W_k$ and aggregates them to update the global model W'.

The algorithm is visualized in a typical application scenario by Google in Figure 1. Scenarios may differ from the one mentioned here, e.g. direct communication and aggregation of updated weights is possible [3]. However, such approaches lie beyond the scope of this paper.

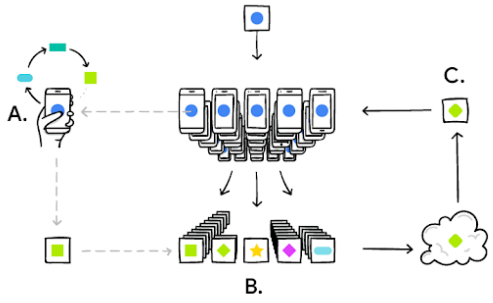


Figure 1: Each client's phone personalizes the model locally, based on their usage (A). Many users' updates are aggregated (B) to form a consensus change (C) to the shared model, after which the procedure is repeated. [4]

2.2. Categorizations of FL Algorithms

Federated learning algorithms can be classified by type of data partitioning, used machine learning model, privacy mechanism, and communications architecture. [2]. Since this paper is focused on accuracy trade off only first two classifications are discussed here.

doi: 10.2313/NET-2022-11-1_04

2.2.1. Data partitioning. Another study proposed categorization of tasks for FL algorithms which is based on data split pattern [3].

Horizontal federated learning (a), also known as sample-based federated learning, is applicable in scenarios where all individual datasets D_k have about the same feature space but are different in samples. FL systems aimed for such scenarios usually have specific architecture in which all clients compute local gradient and communicate it to the server for aggregation into global model. One of the major complications here is that real world datasets are rarely independent and identically distributed (IID), which causes problems for aggregation of local models [1].

Vertical federated learning (b) is applied when feature spaces of datasets D_k overlap only a little, but the sets of IDs overlap significantly. In this scenario various ML algorithms proved to be effective, including statistical analysis, gradient descent and safe linear regression [2].

Federated transfer learning (c) is applied to scenarios when neither feature spaces nor IDs spaces of datasets are close to each other [1]. Such algorithms require complicated architectures and are not reviewed in this paper.

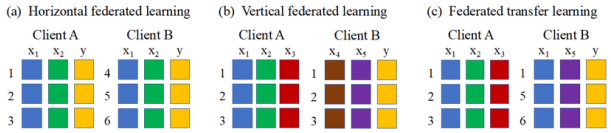


Figure 2: Categorization of Federated Learning by data partitioning. [5]

2.2.2. Applicable Machine Learning model. Three families of ML algorithms to be combined with federated learning approaches can be considered: linear models, tree models, neural networks (NNs) [2]. Each type of ML model can be useful under constrains on performance of clients' devices, data types and distribution of data between clients. However, NNs and some tree boosting algorithms are more popular today [3].

3. Original Federated Learning Algorithms

3.1. Background

In 2017, McMahan et al. introduced the term Federated Learning and created two implementations that formed the basis of the whole branch. Both of the approaches use stochastic gradient descent (SGD), as most of successful applications of deep learning relied on this technique at the time [6]. Minibatch SGD is used as a reference (and baseline for benchmarks) for these algorithms, since it is controlled by hyperparameters that can be easily adopted for distributed ML techniques. The paper proposed that a C-fraction of clients is selected on each round t, and gradient of the loss function over the D_k is computed. Selected clients performs E epochs of local-update SGD with a mini-batch size B; loss function $l(w, D_k)$ is to be minimized. P_k denotes the index set of samples in D_k, n being the total number of samples. This approach is focused on non-convex neural networks objectives, but the architecture can be applied to wider family of ML models. These algorithms are widely used in scenarios of horizontal federated learning [1].

3.2. The algorithms

3.2.1. FedSGD. The Federated Stochastic Gradient Descent (FedSGD) algorithm derives its settings from large-batch synchronous SGD. This algorithm is referred to as a naive approach for the FL algorithm. Proposed implementation with $C = 1$ (i. e. full-batch gradient descent) and fixed learning rate η includes each client k computing averaged gradient on D_k: $g_k = \nabla F_k(w_t)$, w_t being the current model, and then central server updates w_t using aggregated gradients: $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$.

3.2.2. FedAvg. For Federated Averaging (FedAvg) algorithm, more computation is added to each client by iterating the local update multiple times before averaging. Algorithm 1 is an exact reproduction of pseudocode from [1]. Three parameters are controlling the amount of computation and accuracy of the final model: selected fraction of clients C, number of epochs E each data owner goes through his batch to compute local gradient, and local minibatch size B($B = \infty$ indicates usage of the whole local dataset as training dataset); the algorithm with $B = \infty$ and $C = 1$ is the same as FedSGD described earlier.

Algorithm 1 Federated Average, [6]

Server executes:
initialize w_0
for each round $t = 1, 2, ...$ **do**
  $m \leftarrow max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m items)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow$ ClientUpdate (k, w_k)
  end for
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end for

ClientUpdate(k, w)**:**
$\mathcal{B} \leftarrow$ (split P_k into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla l(w, b)$
  end for
  return w to server
end for

4. Further Improvements

Since 2017, when FedAvg was introduced, many attempts to improve this algorithm were made [1].Some of the most established approaches are mentioned in this chapter.

doi: 10.2313/NET-2022-11-1_04

4.1. FedProx

A detailed review of theFederated averaging technique pointed out the importance and influence of number of epochs E on the resulting performance of the algorithm [7]. From obtaining these insights a modification of this algorithm called FedProx was introduced. In this approach each client minimizes approximated loss function instead of exact one. In particular, local updated weights w_{t+1}^k are obtained by solving $\min_w h_k(w, w_t) = l(w, b) + \frac{\mu}{2}||w - w_t||^2$, i.e. minimizing loss function l with given batch b so that $||w - w_t||^2 < \epsilon$. This is a simple way to ensure that local updates are not too far from the global optima. μ is another hyperparameter that has to be tuned; very low values of μ impose almost no regularization effect, and large values cause a drastic slowdown of convergence rate.

4.2. SCAFFOLD

Another study introduced the algorithm SCAFFOLD (Stochastic controlled averaging for federated learning) which aimed to improve performance of Federated averaging in case of non-IID partitioned dataset by using variance reduction techniques [8]. Control variates are introduced both for the server (c) and clients (c_k), and are used to determine corresponding update directions. These variates are updated either by reuse of previously calculated gradients or by computing values of gradients of the local datasets on global model. First approach implements idea similar to Gradient descent with momentum and has lower computation costs, while the second tends to be more stable. This algorithm can significantly improve convergence rate, but communication costs are double the costs of FedAvg because of additional control variates.

4.3. FedNOVA

The Federated normalized averaging (FedNOVA) algorithm is also similar to FedAvg, but has improved aggregation stage. In this algorithm the local updates w_{t+1}^k are scaled in accordance with the actual volume of locally performed training, which may be different because of time constraints combined with different computation speeds of clients, or ranging sizes of local datasets. This allows prioritizing better-trained local weights while keeping communication costs at the same level as FedAvg and only slightly enlarging local computation costs [9].

4.4. Worth noting

Besides algorithms reviewed here, a lot of different federated learning approaches were introduced. Out of this large number of algorithms we considered worth noting the FedPAGE [10], which improves convergence rate in both convex and non-convex settings by implementing recent probabilistic gradient estimator instead of SGD and FedFA [11], which aims at achieving better accuracy and fairness in horizontal federated learning scenarios by employing double momentum gradient and new weight selection algorithm. We consider analyzing accuracy of these and other algorithms as a possibility for future work.

5. Accuracy Analysis of Horizontal Federated Learning Approaches

5.1. FedAvg and FedSGD

5.1.1. Synthetic IID data. Experiments showed that FedSGD and FedAvg can both perform well with a wide range of ML models including multi-layer perceptron, convolutional NNs, two-layer character Long short-term memory (LSTM) networks and large-scale word-level LSTM, and FedAvg tends to reach better test accuracy than FedSGD with same number of communication rounds. To give a better insight of how these FL approaches perform, let us review benchmarks performed on CIFAR-10 [12] with balanced and IID data partitioning [6]. It is shown that FedAvg reaches an accuracy of 85% already after 2000 communication rounds, while centralized SGD needs around 197500 communications to show a similar accuracy of 86%, since communication after every batch is assumed. However, if compared by number of minibatch computations, SGD on united data has better convergence, as gradient updates after each minibatch computation (see Figure 3).

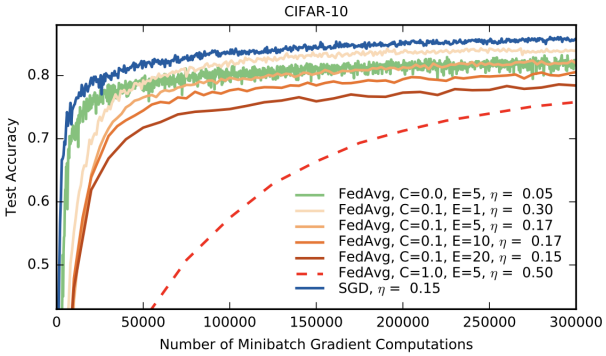


Figure 3: Comparison of accuracy on IID data [6].

5.1.2. Further evaluation. In a later study [13] benchmark aimed specifically for testing performance from different perspectives was developed. It was shown that both these FL approaches converged towards an accuracy level similar to that of a centralized ML model, and significantly higher than that of the same ML model trained only on local data; these results were obtained on IID partitioned MNIST, FEMNIST and ColabA datasets using MLP and LeNet ML models. For robustness evaluation the same datasets were used, but now with non-IID partitioning between clients; it was shown that accuracy of FL approaches drops when clients have only instances of few classes in their training datasets, e. g. in the scenario when training dataset was split so that each client has only one class of samples (see Figure 4).

Theoretical analysis showed [14] that for quadratic objectives performance of FL approaches is strictly better then that of minibatch algorithms, and accelerated variant is minimax optimal. For general convex objectives it is proven that first error upper bound of FL is not worse than that of minibatch SGD, if typical noise scaling is applied. However, lower bound of the error of local SGD

doi: 10.2313/NET-2022-11-1_04

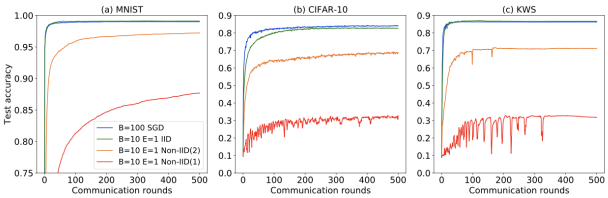
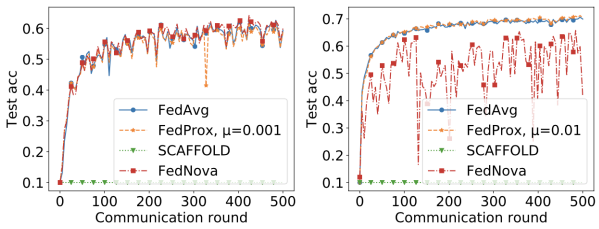


Figure 4: Comparison of accuracy on non-IID data. [13]

approach in worst-case scenario is higher than the worst-case error of minibatch SGD.

5.2. Other Algorithms

Performance of FedAvg, FedNOVA, SCAFFOLD and FedProx was analyzed in the setting of non-IID partitioned dataset [15]. This study showed that none of mentioned algorithms dominates others, and each of them can show outstanding performance under specific constrains. In case of label distribution skew or quantity skew FedProx usually performs better than mentioned alternatives. The accuracy of SCAFFOLD is quite unstable, but in some cases it can significantly outperform other approaches.FedNOVA does not show superiority inn analyzed scenarios, but can sometimes be slightly more effective than other approaches. To illustrate behavior of different approaches, their learning curves on CIFAR-10 [12] with partition in accordance with Dirichlet distribution are shown in Figure 5. We chose this dataset as it was stated to be a challenging task for federated learning approaches under non-IID conditions.



(a) $p_k \sim Dir(0.5)$ (b) $q \sim Dir(0.5)$

Figure 5: Comparison of accuracy on non-IID data from CIFAR-10 with 100 parties, $C = 0.1$, Dirichlet distribution is used to simulate Label distribution skew (left) and Quantity skew (right) [15].

6. Approaches for Vertical Federated Learning

6.1. SecureBoost

A recent study introduced a new algorithm called SecureBoost that is aimed at vertical federated learning [16]. This system implements federated tree-boosting ML model in accordance with principles of FL that is lossless, i.e. is as accurate as other non-federated tree-boosting algorithms trained on centralized data. SecureBoost is based on the XGBoost algorithm, which was recently shown to be one of the most effective ways to work with panel data [17].

6.2. SecureBoost+

Slightly modified version of this algorithm with improvements in performance on large and high dimensional datasets called SecurityBoost+ was recently published [18]. The new approach converges towards similar accuracy, but can be trained much faster to reach the same classification error. The following table illustrates their performance.

Data set	XGB	SecureBoost	SecureBoost+
Give-credit	0.872	0.874	0.873
Susy	0.864	0.873	0.873
Higgs	0.808	0.806	0.8
Epsilon	0.897	0.897	0.894

Figure 6: Area under the receiver operating characteristic (ROC) curve for SecureBoost, SecureBoost+ and XGBoost on centralized data for different datasets [18].

7. Conclusion

In this paper we surveyed the literature on various implementations of federated learning approaches and provided information on existing benchmarks of these algorithms, analyzing their accuracy in different scenarios. We showed that existing implementations can demonstrate very similar accuracy in comparison with centralized approaches while having advantages in terms of privacy, and are guaranteed to reach the same accuracy in some scenarios. However, some settings can impose difficulties for such approaches, e.g. non-IID partitioned datasets held by clients, which is quite common in real-world problems. Depending on the scenario different approaches reviewed in this article can show better performance, and none of them can be considered as universally best one.

References

[1] H. B. McMahan *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1, 2021.

[2] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[4] "Federated learning: Collaborative machine learning without centralized training data," Apr 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[5] S. Chen, D. Xue, G. Chuai, Q. Yang, and Q. Liu, "Fl-qsar: a federated learning based qsar prototype for collaborative drug discovery," 02 2020.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[7] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, vol. 3, p. 3, 2018.

doi: 10.2313/NET-2022-11-1_04

[8] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.

[9] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[10] H. Zhao, Z. Li, and P. Richtárik, "Fedpage: A fast local stochastic gradient method for communication-efficient federated learning," *arXiv preprint arXiv:2108.04755*, 2021.

[11] W. Huang, T. Li, D. Wang, S. Du, and J. Zhang, "Fairness and accuracy in federated learning," *arXiv preprint arXiv:2012.10069*, 2020.

[12] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[13] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018. [Online]. Available: https://arxiv.org/abs/1806.00582

[14] B. Woodworth, K. K. Patel, S. Stich, Z. Dai, B. Bullins, B. Mcmahan, O. Shamir, and N. Srebro, "Is local sgd better than minibatch sgd?" in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 334–10 343.

[15] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," *arXiv preprint arXiv:2102.02079*, 2021.

[16] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[17] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[18] W. Chen, G. Ma, T. Fan, Y. Kang, Q. Xu, and Q. Yang, "Secureboost+: A high performance gradient boosting tree framework for large scale vertical federated learning," *arXiv preprint arXiv:2110.10927*, 2021.

Survey on Scheduling Approaches in TSN

Leon Kist, Philippe Buschmann*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: leon.kist@tum.de, phil.buschmann@tum.de

***Abstract*—Modern concepts such as Industry 4.0 or in-vehicular networks feature complex, interconnected systems. These systems often depend on highly time-critical communication. One approach to provide this type of communication is supplied by Time-Sensitive Networking (TSN). TSN is standardized in IEEE 802.1Q and encapsulates several different mechanisms. Among these mechanisms is the Time-Aware Shaper (TAS), which allows the creation of Time Division Multiple Access (TDMA) schemes for traffic in the network. However, the synthesis of these schemes entails several problems, such as exponential computation times. Another difficulty arises from the prioritization of traffic, i.e., which streams to prioritize in order to maximize the functionality of applications.**

In this paper we survey two different approaches to scheduling in TSN. The first approach leverages hierarchical structures in factory automation networks to simplify schedule creation. The second approach aims at improving the Quality of Service (QoS) of less time-critical traffic, while still adhering to the requirements of more time-critical traffic. Both approaches succeed in their goals.

***Index Terms*—time-sensitive networking, scheduling**

1. Introduction

Modern concepts such as Industry 4.0, smart factories, and in-vehicular networks feature complex, interconnected systems which often depend on various IoT devices and sensors to enhance productivity and avoid errors or malfunctions [1] [2]. In order to provide such functionality, communication between different elements of these systems, needs to adhere to strict timing constraints or even has to have real-time properties [3].

One approach to meet these constraints is provided by *Time-Sensitive Networking* (TSN). TSN uses wired ethernet and is standardized by the IEEE TSN task group, which has released several standards for TSN starting in 2012. One such standard, IEEE 802.1Qbv (now included in IEEE 802.1Q), introduces the *Time-Aware Shaper* (TAS). The TAS can be used to create a *Time Division Multiple Access* (TDMA) scheme for traffic in the network, based on a given schedule [4] [5] [3] [6].

Schedule creation for the Time-Aware Shaper however, entails a number of different problems. The creation of schedules itself can be considered as a NP-Hard problem. The computational effort associated with creating schedules is therefore often large. This is especially true in many common applications where networks are large and many streams of data are exchanged. Another issue is the prioritization of different streams in the network. Some components can tolerate missing communication deadlines, while others would fail in the presence of a single deadline-miss. In order to maximize the functionality of applications it is necessary to avoid deadline-misses for highly critical traffic, while still providing adequate properties for less critical streams [3] [6].

The goal of this work is to survey different approaches or improvements to scheduling for the *Time-Aware Shaper* in TSN. To do this we will first explore some related work in Chapter 2. Afterwards in Chapter 3 some essential background information will be explained. Next, we will analyze the scheduling approaches in Chapter 4. After this in Chapter 5 we will compare the approaches. Then finally Chapter 6 will provide some conclusions and a possible outlook for future work.

2. Related work

As this is a survey of different, preexisting works, we will explore some of the mentioned works in more depth in Chapter 5. For now, we only give a short outline of the main papers we are surveying.

In the first work we analyze Hellmanns et al. [3] describe an hierarchical scheduling approach that exploits common properties of Factory Automation Networks.

Second we explore a paper by Houtan et al. [6], which seeks to improve existing scheduling approaches in regards to the performance of less time critical best effort (BE) traffic.

While there exist numerous other papers on scheduling approaches or improvements thereof, we are not able to analyze them further, due to the limited scope of this work. Other work includes an approach by Oliver et al. [7], which focuses on the creation of schedules using the first-order theory of arrays. Another paper by Vlk et al. [8], seeks to improve the performance of scheduled traffic, as well as the creation of schedules by improving hardware and the scheduling mechanism itself. A paper by Syed et al., explores a Mixed-integer Programming based approach for both scheduling and routing combined [9].

3. Background

In this chapter we provide some background information necessary for understanding the surveyed approaches.

doi: 10.2313/NET-2022-11-1_05

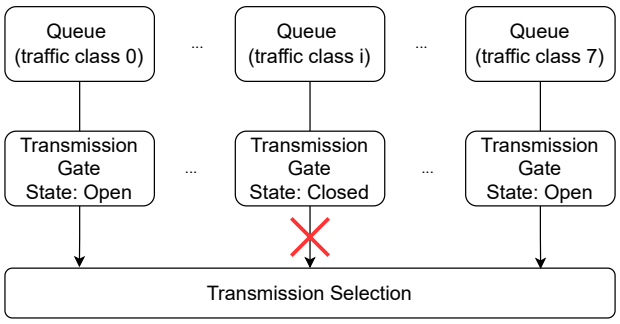


Figure 1: Simplified visualization of transmission selection (based on Figure 8-14 of IEEE 802.1Q) [4]

3.1. The Time-Aware Shaper

As already mentioned, the TAS can be used to create TDMA-schemes for traffic in a network. To achieve this, IEEE standard 802.1Qbv specifies a gating procedure to restrict the amount of traffic forwardable by a switch at each point in time [3] [6].

To understand this mechanism, it is necessary to first explore transmission selection in TSN capable switches (as specified by IEEE 802.1Q). Such switches possess between 1 and 8 separate queues for each of its egress ports (transmission selection happens separately for each egress port). Each queue corresponds to a traffic class. Traffic classes are categorizations assigned to frames/streams based on their priority. Frames are chosen for transmission based on traffic class and transmission selection algorithms supported by the queues [4].

The gating mechanism introduced by IEEE 802.1Qbv now specifies an additional *Transmission Gate* for each queue. This gate can either be *Open* or *Closed*. If the *Transmission Gate* for a queue is closed, then frames of that queue are ineligible for transmission selection. A simplified version of this is illustrated in Figure 1.

The state for each gate is controlled by the so called *Gate Control List* (GCL). The GCL is an ordered list of *Gate Operations*. Each *Gate Operation* consists of a vector specifying the state for all gates and a time interval specifying how long the state should persist. The sum of all such time intervals is referred to as *network cycle*. All egress ports cycle over a separate GCL. Schedule synthesis effectively corresponds to the creation of *Gate Control Lists*. Suitable timing allows not only for scheduling on traffic class level, but also on stream-level [5] [3].

3.2. Schedule Creation

Approaches for schedule creation can generally be distinguished into two distinct groups: *exact approaches* and *heuristic approaches Exact approaches* calculate solutions with provable properties, such as optimality or satisfiability. These approaches are often based on mathematical methods, like *Integer Linear Programming*, *Satisfiability Modulo Theories* (SMT), or *Optimization Modulo Theories* (OMT) [3] [6].

While the calculation of exact results can be advantageous, it is generally a NP-Hard problem. Using mathematical methods can therefore cause issues, such as very high resource consumption or exponentially long computation times. This is especially problematic when calculating schedules for larger networks [3] [6].

To avoid these problems, it is also possible to use *heuristic approaches*. Heuristic approaches forgo the calculation of provably optimal solutions in favor of an optimized calculation duration [3] [6].

4. Analysis

In this chapter we analyze the previously mentioned surveyed work. First, we explore the hierarchical approach proposed by Hellmans et. al [3]. After this we dive into improvements to the performance of Best-effort (BE) traffic introduced by Houtan et al. [6].

4.1. Hierarchical Approach

As already noted, the creation of schedules can be computationally intensive. This is especially true for large networks or networks where a large amount of different data-flows are exchanged. In order to reduce this overhead, Hellmans et al. propose a scheduling approach leveraging the hierarchical structure of Factory Automation Networks [3].

4.1.1. Terminology and Network Characteristics. Before we elaborate on the actual scheduling approach proposed in [3], we need to explain some specific terminology and assumptions about the network.

The first term we define is *stream isolation*. This refers to scheduling where each stream is assigned a specific (separate) time slot in the created TDMA-scheme. In contrast, *stream batching* refers to scheduling where multiple streams share a single time slot.

Next we briefly explain certain characteristics of *factory automation networks* as used in e.g., smart factories [10]. Such networks are hierarchical in nature and can therefore be divided into several sub-networks. These sub-networks are connected by *boundary switches*.

Based on [11] and [3] we differentiate between several types of traffic, each with unique deadline/latency requirements. However, to understand this work it is mostly important to understand that such types exist. The only type we need to elaborate on in a more detailed manner is *isochronous traffic*. This type of traffic has strict deadline requirements and occurs periodically. The exact timing of when an isochronous stream is sent is chosen during design time. All other traffic types have less strict timing requirements.

We can distinguish two types of streams of isochronous traffic: *intra-level streams* (within a sub-network) and *inter-level streams* (between sub-networks). We assume that *intra-level streams* have even stricter timing constraints than *inter-level streams* [3].

4.1.2. Approach. Now that the necessary terminology and network characteristics have been explained, we can explore the scheduling approach. This approach envelopes two separate parts: (1) a *phase model* which isolates the different types of traffic in the network and (2) a *schedule creation* mechanism for isochronous traffic [3].

The **phase model** essentially divides the *network cycle* into separate *cycle phases*. Each *cycle phase* is assigned to

doi: 10.2313/NET-2022-11-1_05

a traffic type. This *cycle phase* is then exclusively reserved to this single traffic type. Scheduling happens separately for each *cycle phase*. This simplifies scheduling, as schedulers only ever have to consider one type of traffic at a time. It also allows the usage of different scheduling mechanisms for different types of traffic [3].

The **schedule creation** mechanism for isochronous traffic consists of two stages executed in the following order: 1) intra-level scheduling (intra LS) and 2) inter-level scheduling (inter LS). Separating the traffic in such a manner is possible due to the hierarchical nature of factory automation networks, as well as the differences in timing constraints between intra-level streams and inter-level streams. Both phases are visualized in Figure 2.

Stage 1): *Intra LS* creates a separate schedule for each *machine ring* sub-network. This drastically reduces the complexity of schedule synthesis. Schedules incorporate both intra-level streams and inter-level streams. However, inter-level streams are only forwarded to their respective boundary router (on the local ring) in this stage. There they are queued until the second stage. *Intra LS* uses *stream-isolation* to accommodate the strict timing requirements of intra-level streams. Any of the previously mentioned schedule creation approaches (e.g., heuristic, SMT, OMT) can be used in this stage.

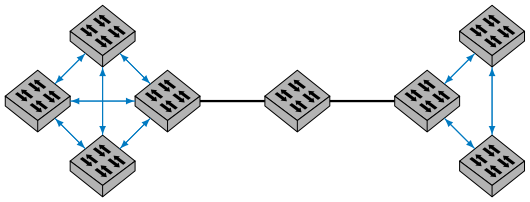
Stage 2): *Inter LS* is used to create schedules for inter-level streams (which are buffered at their boundary routers after the first stage). However, in contrast to stage 1), *stream-batching* is used instead of *stream-isolation*. All streams queued at a boundary router after stage 1) are grouped together. Schedule creation itself does not use any of the previously mentioned approaches. Instead, the forwarding of all isochronous traffic is simulated, as if there was no scheduling (i.e., all gates for isochronous traffic are open). The simulation then determines for how long each gate has to be open. After all gates are closed the *cycle phase* for isochronous traffic can end. As there is no other traffic in the network during stage 2) (due to the *phase model*), the simulation should be able to accurately predict forwarding in the network.

It is important to note, that should stage 2) detect a deadline failure, it has few mechanisms to prevent this. The only possible recourse is to try and manipulate the arrival time of said stream at the boundary switch in stage 1) to gain an advantageous position in the queue of the switch. However, as it is assumed that inter-level streams are less time-critical, no detailed strategy is presented [3].

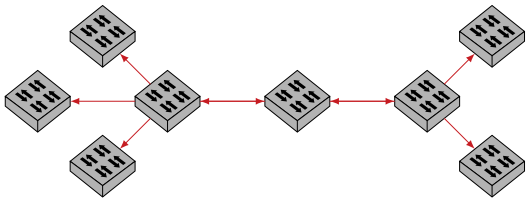
4.2. QoS Improvements

As already mentioned, TSN can be used to guarantee the timing requirements of time-critical traffic in networks. However most networks also feature less (or non) time-critical traffic. While delaying this traffic might not be as detrimental to the application, as delaying highly time-critical traffic, it can still affect its performance negatively. This approach therefore aims to improve the Quality-of-Service (QoS) of less time-critical BE traffic, while still guaranteeing the deadlines of time-critical streams [6].

4.2.1. Terminology and OMT. Before we can explore the actual approach, we once again need to define some terminology.



(a) Phase 1): *Intra LS*



(b) Phase 2): *Inter LS*

Figure 2: Visualization of phases in the heuristic approach for a simple network

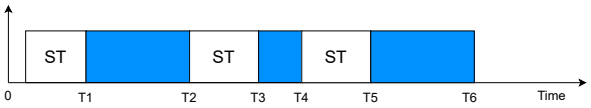


Figure 3: Visualization of slack (in blue) based on [6]

As in the previous approach we need to differentiate between multiple types of traffic in the network. However, for this approach we only need to distinguish highly time-critical scheduled traffic (ST) and less time-critical BE traffic.

This approach uses the previously mentioned OMT, and can therefore be classified as an exact approach. OMT allows the scheduling problem to be expressed as a constrained optimization problem. This problem can then be solved using SMT/OMT solvers [6]. The approach seeks to improve a previous approach by Craciunas et al. proposed in [12].

4.2.2. Approach. To accommodate BE frames, the concept of *slack* is introduced. *Slack* describes a time interval after a ST frame is sent in which no other ST frame can be sent. This time may then be used for BE frames. This is visualized in Figure 3 This approach essentially aims to find feasible schedules for ST traffic, while still guaranteeing a certain *slack*. In order to achieve this, several new constraints and objective functions are introduced. As the scope of this work is quite limited, we will only explain their objective and effects. For a more detailed view, we refer to [6].

To optimize QoS of BE traffic, the approach introduces four new **constraints**.

The *porous link constraint* is a modification of the *link constraint* introduced in [12]. The original constraint had the purpose of preventing temporal overlap of frames on the same link, i.e., two frames being sent at the same time. This modified version also incorporates *slack*, it therefore

doi: 10.2313/NET-2022-11-1_05

disallows overlap of ST frames with previous ST frames and their associated slack.

The *slack size constraint* has the purpose of defining the acceptable *slack* size for each frame.

The *hop slacks constraint* encompasses several equations, which ensure that the total amount of *slack* on a link is within acceptable bounds.

The *equal slack constraint* is optional. Its purpose is to enforce equal slack sizes for all frames on one link. This can be used to improve the optimization time, as it limits the amount of eligible ST schedules [6].

In addition to these new constraints, three new **objective functions** are introduced.

In previous work a *minimization* objective function is used. This function minimizes the total offset of ST streams. Therefore, schedules created using this function (subject to the preexisting constraints mentioned), feature all ST traffic right in its beginning. This can be detrimental to BE traffic in the network.

The first objective function introduced is the *maximization* function. In contrast to the *minimization* function, the *maximization* function, puts all ST frame transmissions as close to their deadlines as possible. All deadlines are therefore still guaranteed, however BE traffic can (depending on the amount of ST frames) be scheduled much earlier (in many cases). This function does not use the notion of *slack* and is therefore still only subject to the preexisting constraints.

The *sparse schedule* objective function aims to maximize the total amount of slack on each link. Schedules created by this function distribute ST frames sparsely, with gaps (*slack*) in between for BE traffic. This function is subject to both the preexisting and newly introduced constraints (without the *equal slack constraint*).

The *evenly sparse schedule* objective function is essentially the same as the *sparse schedule* objective function, however it is also subject to the *equal slack constraint*. This leads to the creation of schedules with an even distribution of ST frames. Gaps (*slacks*) between ST frames are of the same length.

Using the *sparse schedule* function or the *evenly sparse schedule* function has the advantage of avoiding long queueing times for BE frames, as well as reducing the search space, which can lead to faster computation times [6].

5. Evaluation

This chapter is dedicated to the evaluation of the introduced approaches. First, we discuss the results achieved by the two approaches. Afterwards we assess their compatibility.

5.1. Results

In this section we present and compare the results measured in both papers.

5.1.1. Hierarchical Approach. To evaluate the effectiveness of their approach, Hellmans et al. compare the performance of their approach to that of a (heuristic) global 1-stage approach. The same approach is used for *intra LS* in the hierarchical approach. The following metrics are evaluated: execution time, cycle phase duration, and the average number of GCL entries per port. Each metric is evaluated for a variety of topologies using different amounts of sub-networks and sub-network sizes. The amount of streams in the network is varied as well.

The impact of different topologies is minor compared to that of the number of streams in the network. Regarding *cycle phase length*, both approaches perform about the same with slightly shorter (<0.1ms) cycle phases using the global approach. The *amount of GCL entries* is comparable for both approaches when fewer streams are in the network ($\leq$500). When more streams are present, the hierarchical approach significantly outperforms the global approach, up to a factor of four. This is because the global approach requires two entries per hop, per stream. The hierarchical approach acts similarly in stage 1), but only requires two additional entries in stage 2). The largest difference occurs when comparing the *execution times*. Similarly to the GCL lengths, the execution times are comparable for fewer streams ($\leq$100), with the global approach even outperforming the hierarchical approach. This is due to the overhead caused by the increased number of calls to the scheduler in stage 1. However, for more streams the global approach is outperformed up to a factor of more than 100. For more than 1250 streams, the global approach is unable to find a solution (even after more than 200h), due to insufficient hardware capabilities [3].

5.1.2. QoS Improvements. To evaluate their results, Houtan et al. simulate a network with six end stations, two connected TSN switches and ten streams. Streams have varying probabilities to be ST or BE. To measure the QoS of BE streams, the end-to-end delay of BE streams, deadline misses (for BE streams), as well as execution time (of schedule synthesis) were chosen.

Regarding end-to-end delay, the *minimization* function performs the worst out of all four functions. The *sparse schedule* function and *evenly sparse schedule* function perform about the same. The *maximization* function performs the best out of all objective functions. This remains true across all scenarios. The *maximization* function outperforms the other functions, since it schedules ST traffic as late as possible. This allows BE traffic to be scheduled earlier, thereby decreasing its end-to-end delay. Weaknesses of the *maximization* function reveal themselves when looking at deadline misses. Here the *maximization* function, as well as the *minimization* function produced significant amounts of deadline misses for one of the tested scenarios. Both the *sparse schedule* function and *evenly sparse schedule* function produce virtually no deadline misses in any of the scenarios. The *sparse schedule* function and *evenly sparse schedule* function also perform the best regarding execution time. The *minimization* function performs significantly worse than either of the previously mentioned functions. The performance of the *maximization* function varies depending on the amount of ST streams in the network. The more ST streams are present, the worse its performance becomes. In the presence of large amounts of ST streams, the *maximization* function is significantly outperformed by the *minimization* function [6].

5.1.3. Comparison. Both approaches have quite different objectives and mechanisms. This, in conjunction with the vastly different network topologies used for their measurements, make a comparison between the results of the approaches quite difficult. The only metric measured for both approaches is execution time. Both approaches succeed in significantly improving the schedule creation time, under the right circumstances. For the heuristic approach, these improvements depend on the amount of streams in the network. Only in the presence of more than 100 streams, does the hierarchical approach provide significant improvements. However, the more streams are present, the larger the improvements. The global approach in turn is only measured for a fairly small network with few streams. However here it manages to significantly improve the execution time [3] [6].

We can therefore conclude that for small networks or networks with few streams ($\leq$100) the global approach would yield larger performance improvements. For networks with a great number of streams (>100) however, the hierarchical approach is likely to achieve greater improvements. In networks with more than 1250 streams it is possible that a hierarchical approach is the only feasible solution (depending on the existing hardware).

5.2. Compatibility

The first scheduling approach introduced divides the *network cycle* into several different phases for each type of traffic. Scheduling is done for each type of traffic separately. In contrast, the second approach aims at finding a unified schedule for all traffic combined. As the first approach requires separation of traffic types and the second approach requires knowledge about all traffic, a direct combination of both approaches seems infeasible. However the notion of leveraging hierarchical structures in the network itself and using a multi-stage approach could be used for multiple types of traffic combined. In a setup like this, something akin to the second approach might be applicable.

6. Conclusion and Future Work

In this survey we look at two very different approaches to improve scheduling for TSN. The first approach we survey aims to utilize hierarchical characteristics of the network to improve scheduling. The second approach tries to improve the performance of less time-critical traffic by better utilizing the deadlines of more critical traffic. Both approaches succeed in improving different aspects of scheduling.

Looking at the hierarchical approach, we are able to see that leveraging characteristics of certain types of networks allows for significant improvements to scheduling. Determining and utilizing such characteristics could provide an excellent avenue for future work.

Another interesting approach might be to combine the introduced hierarchical, multi-stage approach with other approaches in order to explore their compatibility.

From the improvements to QoS of less time-critical traffic we can see that synthesizing feasible schedules for highly time-critical traffic is possible, even in the presence of other objectives. It might therefore also be worth investigating if there are issues with other scheduling approaches which can be mitigated or solved like this.

References

[1] "What is Industry 4.0?" https://www.ibm.com/topics/industry-4-0, [Online; accessed 21-March-2022].

[2] W. Zhou and Z. Li, "Implementation and Evaluation of SMT-based Real-time Communication Scheduling for IEEE 802.1Qbv in Next-generation in-vehicle network," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 457–461.

[3] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling TSN Scheduling for Factory Automation Networks," in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020, pp. 1–8.

[4] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

[5] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[6] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Synthesising Schedules to Improve QoS of Best-Effort Traffic in TSN Networks," in *29th International Conference on Real-Time Networks and Systems*, ser. RTNS'2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 68–77. [Online]. Available: https://doi.org/10.1145/3453417.3453423

[7] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 13–24.

[8] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing Schedulability and Throughput of Time-Triggered Traffic in IEEE 802.1Qbv Time-Sensitive Networks," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 7023–7038, 2020.

[9] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "MIP-based Joint Scheduling and Routing with Load Balancing for TSN based In-vehicle Networks," in *2020 IEEE Vehicular Networking Conference (VNC)*, 2020, pp. 1–7.

[10] "What is Factory Automation and what are the benefits for today's IoT enterprises?" https://www.cassianetworks.com/blog/factoryautomation/, [Online; accessed 15-May-2022].

[11] "Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types," https://hub.iiconsortium.org/tsn-converged-traffic-types, [Online; accessed 29-March-2022].

[12] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 183–192. [Online]. Available: https://doi.org/10.1145/2997465.2997470

A Short Introduction To MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer

Florian Raabe, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: florian.raabe@tum.de, christopher.harth-kitzerow@tum.de

***Abstract*—The MASCOT protocol allows a secure multiparty computation of arithmetic circuits over a finite field. Using oblivious transfer in an arithmetic context, it creates multiplication triples which are used to compute products of additively secret-shared values. The expensive computation to securely generate these triples is done in a preprocessing phase. After a one-time setup, the protocol is based entirely on fast, symmetric cryptography. By making use of efficient oblivious transfer extensions, the total cost for multiplications is reduced. With careful consistency checks and other techniques for privacy amplification MASCOT achieves active security. It considers a dishonest majority where any number of parties can act actively maliciously.**

***Index Terms*—multiparty computation; oblivious transfer**

1. Introduction

Secure Multiparty Computation (MPC) is a cryptographic method to jointly evaluate a function on private inputs without revealing those to the other parties. To realize this functionality, one approach is to secret-share input values between all parties. Most protocols use a linear secret-sharing scheme which allows the local addition and subtraction of shares. The parties obtain a share of the corresponding operation on the secrets. A simple example is the additive secret-sharing scheme. To secret-share a value s between n parties, a tuple $(s_1, ..., s_n)$ is sampled uniformly random so that $s_1 + ... + s_n = s$. In order to compute every function in a field on these secret shares, multiplication is needed.

The **M**alicious **A**rithmetic **S**ecure **C**omputation with **O**blivious **T**ransfer protocol by Marcel Keller, Emmanuela Orsini and Peter Scholl [1] also makes use of additive sharing. It allows for efficient and secure computation of general arithmetic circuits using almost exclusively fast, symmetric cryptography. The advantage of arithmetic circuits over Boolean circuits is that secure addition can be done locally, thus not requiring any communication.

MASCOT is the first protocol to use oblivious transfer with a dishonest majority setting to generate multiplication triples in any sufficiently large field. It works with n parties and considers a corruption of up to $n - 1$ active malicious adversaries. The adversary is considered to be static, meaning that corruption can only take place before a protocol starts.

The MASCOT protocol achieves this through simple consistency checks and privacy amplification techniques which will be introduced in the following section.

2. Preliminaries

The main task in preparing the MPC protocol is the creation of multiplication triples. These are additive secret sharings of tuples $(a, b, a \cdot b, a \cdot \Delta, b \cdot \Delta, a \cdot b \cdot \Delta)$ where a, b are random values used for the multiplication and Δ is a secret-shared random global MAC key. To generate a triple, the shares for a, b and Δ can be chosen randomly by each party. How secret sharings of the products are created will be shown in Section 3.

2.1. Information-theoretic MACs

Message authentication codes (MACs) are short tags used to confirm the authenticity of a message or its sender. For this, strong universal functions can be used. In this context, information-theoretic refers to the security aspect of the MAC. Perfect security can never be achieved since the adversary can always guess a random tag. This is why the probability to find a valid tag should be $\frac{1}{2^{|n|}}$ for n-bit fields [2]. In this protocol, a secret value x is represented by

$$[\![x]\!] = (x^{(1)}, ..., x^{(n)}, m^{(1)}, ..., m^{(n)}, \Delta^{(1)}, ..., \Delta^{(n)}).$$

Each party P_i holds a random share $x^{(i)}$, a random MAC share $m^{(i)}$ and a share of the fixed MAC key $\Delta^{(i)}$, such that the MAC relation $m = x \cdot \Delta$ holds. $[\![.]\!]$ denotes the linear authenticated secret sharing scheme. To open a value, all parties broadcast their shares to one party which adds them together and publishes the result x. All parties then check the MAC by committing and opening $m^{(i)} - x \cdot \Delta^{(i)}$. These shares then need to sum up to zero in order for the check to pass. To increase efficiency, random linear combinations of the MACs can also be checked.

2.2. Oblivious Transfer

1-out-of-2 Oblivious Transfer (OT) is a protocol between two parties. The sender transmits two messages from which only one will be received. This is decided by the receiver with a choice bit. However, it remains oblivious to the sender which message was received.

doi: 10.2313/NET-2022-11-1_06

Most existing protocols require public-key cryptography to implement this functionality. The MASCOT protocol uses the concept of OT extensions from Beaver introduced in 1996 [3]. A single oblivious transfer is used in combination with a seed as an initialization. From this point on many OTs can be generated with cheap, symmetric primitives. New correlated values can be created by using a generator function and adjusting the output of the receiver. In MASCOT, this is realized with the COPE protocol which will be explained later in Section 3.1. A consistency check in form of a sacrifice is used to make it maliciously secure.

2.3. Sacrificing technique

To ensure the correctness of a secret-shared value, a correlated shared value can be used. The share of a that should be checked is masked with the other secret-shared value $\hat{a}$ by computing $p = s \cdot a - \hat{a}$ where s is a random. The resulting share p is then opened and can be checked by all parties using their part of the global MAC key. To check a triple, another share is computed shown in 1 which has to open to zero. Note that $c = a \cdot b$ and $\hat{c}$ is the correlated value. By this, some bits of the correlated value might be leaked if the adversary input inconsistent values to one of the OTs.

$$s \cdot c^{(i)} - \hat{c}^{(i)} - b^{(i)} \cdot p \tag{1}$$

3. Preparation Phase

To obtain an actively secure product-sharing protocol, MASCOT improves the passively secure protocol of Gilboa [4]. The basic concept of this is to run OT instances between every pair of parties so that every party has a share of the products in the triple. A k-bit field element is split into bits, hence k oblivious transfers are used. Still, malicious parties can provide inconsistent inputs which will lead to potentially incorrect results when the generated triples are used. In order to prevent this, two strategies are used.

First of all, the correctness of the products in the MAC generation has to be ensured. Therefore, random linear combinations of the MACs are checked immediately after the creation as well as later when opening values. Secondly, to verify the correctness of the multiplication triples a standard sacrifice technique is used where a pair of triples is checked in order to use one securely. Furthermore, the privacy of a triple can be assured by producing several triples and taking random combinations to get a uniformly random triple. These strategies are realized by differnet subprotocols shown in Figure 1.

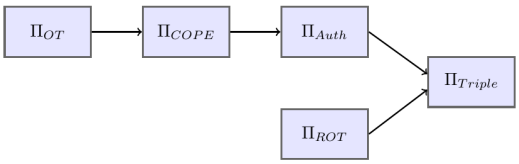


Figure 1: Dependency among subprotocols

3.1. Correlated Oblivious Product Evaluation

To obtain an additive sharing, the MASCOT protocol uses an arithmetic generalization of the passively secure OT extension of Ishai et al. [5]. The correlated oblivious product evaluation (COPE) transforms the multiplication $x \cdot \Delta$ where Δ is fixed at the start of the protocol and future iterations can create sharings for different values of x. The foundation of the COPE protocol is Gilboa's method for oblivious product evaluation.

Oblivious Product Evaluation. The concept of oblivious product evaluation (OPE) uses k sets of oblivious transfers on k-bit strings to obtain an additive sharing of the product. Let us assume P_A is the sender and P_B is the receiver. Now, P_A samples a random value t_i in each iteration and inputs the correlated value $t_i + a$ where a is the sender's input. In every OT the receiver P_B inputs one bit of their secret value. From this follows the output P_B receives in every ith iteration: $q_i = t_i + b_i * a$. Finally, both parties compute the inner product of their values $(q_i)_i$ and $(-t_i)_i$ to obtain q and t for which it holds that $q + t = x \cdot \Delta$.

COPE. The MASCOT protocol now optimizes this functionality of OPE to perform the OTs only once. Therefore, the COPE protocol is initialized at the beginning by calling the **Initialize** command. Because one party's input is still fixed, the receiver simply inputs their bits of Δ. On the other hand, the sender now does not input their secret but k pairs of random λ-bit seeds. In this case, λ is the computational security parameter and $k = \log\lfloor|\mathbb{F}|\rfloor$ the number of bits in the field.

Secondly, the protocol provides the **Extend** command which expands the original seed using a pseudo random function (PRF). This creates k bits of new, random OTs while still remaining the same receiver choice bits. To realize this, P_A uses both seeds in combination with a counter as input for the PRF to create new random seeds. Now a correlation between these outputs is created using the secret input of the sender. This masked correlation is sent to P_B who uses it to adjust their PRF output. Finally, both parties have k correlated OTs on field elements. These are then mapped into a single field element to obtain an additive sharing again.

If both parties follow the protocol, they gain an additive sharing of the multiplication. However, because the MASCOT protocol assures security against an active adversary, it has to be considered what happens if parties do not follow the protocol. Since the input of the receiving party is fixed at the start of the protocol and P_B sends no messages afterwards, there is no possibility to deviate from the protocol. Nevertheless, the sending party might use different input values in the extend phase. This is not considered a security issue because the seeds are uniformly random and the input will later be checked.

doi: 10.2313/NET-2022-11-1_06

3.2. Authentication with COPE

As shown in Figure 1, the functionality of the COPE protocol is used to create authenticated shares. Furthermore, it is used to securely open linear combinations with a MAC checking procedure. The main goal is to prevent the adversary from inputting errors in COPE and opening an authenticated share to the incorrect value. The protocol maintains a dictionary of the authenticated values and includes five commands shown in Figure 2.

Input: takes a list of values $x_1, ..., x_l$ from one party and stores them with identifiers.
LinComb: computes linear functions on values that have been input.
Open: reassembles a secret-shared value and outputs it to all parties.
Check: verifies the correctness of a value that was output by an adversary.
Abort: terminates the protocol and informs all parties, that it failed.

Figure 2: List of commands

In this functionality, the value which was opened might be incorrect. The **Check** command will confirm this. To check a MAC, the party P_i inputs an opened value y, a MAC share $m^{(i)}$ and a MAC key share $\Delta^{(i)}$. Next, they compute $\sigma = m^{(i)} - y \cdot \Delta^{(i)}$, commit and open it. If $\sigma^{(1)} + ... + \sigma^{(n)} = 0$ they continue and the opened value is correct. Otherwise, they abort.

Using the correlated oblivious product evaluation protocol to create authenticated sharings is not enough to ensure active security. To simplify, let us consider a model with only two parties P_1 and P_2. In this scenario, P_1 is honest and wants to authenticate its input x. Therefore, they initialize the COPE protocol and P_1 inputs x in the extend phase. P_2 inputs its MAC key share Δ_2. They receive t and q so that $t + q = x \cdot \Delta_2$. Following both parties define MAC shares $m_1 = x \cdot \Delta_1 + t$ and $m_2 = q$ so that clearly $m_1 + m_2 = x \cdot \Delta$. To create shares of x, P_1 simply generates random additive shares and sends one to P_2. Because the shares and the MACs are linear, both parties can compute linear combinations on authenticated values locally.

Although this is passively secure, if P_1 is actively malicious, it can choose what value to open at the time of opening a value. The party is not committed to opening a particular value and therefore it is not a secure realization of the functionality. To solve this problem, it suffices to authenticate another random value and check a random linear combination of all MACs during the input phase. This requires two changes in the Input stage. P_1 samples a random dummy input x_0 and authenticates it with the other inputs. In addition to this, after computing the MACs using the COPE protocol, P_1 opens a random linear combination of the inputs $x_0, ..., x_l$ and the MAC is checked by all parties. x_0 masks the actual inputs. Hence, P_1 cannot later open to a different value and is committed to their inputs during the Input stage.

Even though the secret values are masked, only random combinations of inputs can be checked. This could be used as an advantage because the check just relates to the randomly weighted sum of the vectors. With a probability of $\frac{1}{|\mathbb{F}|}$ there is one bit in the input vector that does not affect the MAC check. This results in two different vectors and the adversary could decide later which value to open. This can be neglected depending on the security parameter in which the subtrahend comes from the number of possible pairs where the bits are different. The protocol still securely implements the functionality with a statistical security parameter of $\log |\mathbb{F}| - 2 \log \log |\mathbb{F}|$. Note that a repeated check can ensure statistical security of $\log |\mathbb{F}|$.

Finally, the protocol can easily be extended to the use with n parties. When a party P_j inputs a value, they run COPE with every other party $P_i \neq P_j$. Naturally, they provide their MAC key share $\Delta_{(i)}$ as input. This allows P_j to obtain an authenticated share under the global MAC key $\Delta = \Delta_1 + ... + \Delta_n$. Through this, more possibilities emerge where a corrupted party might cheat and deviate from the protocol. They could for example provide inconsistent x's or use an incorrect share of Δ when authenticating other parties input. This is not problematic because except with a probability of $\frac{1}{|\mathbb{F}|}$ the MAC check will fail in the Input stage if this happens.

3.3. Generation of Multiplication Triples

The functionality described before is now being used to generate multiplication triples. In more detail, a triple $(\mathbf{a}, b, \mathbf{c})$ with $b \in \mathbb{F}$ and $\mathbf{a}, \mathbf{c} \in \mathbb{F}^\tau$ will be created. For k-bit statistical security in a k-bit field, it is sufficient to use $\tau = 4$. Note that $\tau = 3$ suffices for $\frac{k}{2}$-bit statistical security. This is due to the probability of passing the sacrifice check and the probability of distinguishing the output distribution from random. By multiplying these the number of triples that have to be combined can be determined to implement the protocol with the according statistical security parameter.

To guarantee the randomness of b, it can be checked with a sacrifice. However, this may leak some bits of $\mathbf{a}$ if a malicious party used inconsistent inputs in some of the OTs before. This is the reason why inner products are used. All parties sample a random value $r \in \mathbb{F}^\tau$ and obtain the triple (a, b, c) with $a = \langle a, r \rangle$ and $c = \langle c, r \rangle$. This ensures that any leaking bits of $\mathbf{a}$ are combined with not leaking bits so a appears uniformly random. The same applies to c. Since b is checked with a sacrifice, a second triple needs to be generated. Instead of repeating this step, another random r can be sampled and used to create a correlated triple with the same b.

The triple generation protocol realizes this optimized idea. It starts with the **Multiply** step which uses random oblivious transfer (ROT) to compute a secret sharing of the product $a \cdot b$. This is done by each pair of parties running τ copies of the basic two-party product sharing protocol. To clarify, for each finally created triple there are τ triples generated which will be combined to one.

doi: 10.2313/NET-2022-11-1_06

In this step a corrupt party might guess some bits of a, that is why τ components of a are used instead of only one. Since b is already uniformly random, no privacy amplification is needed here. Afterwards, each party sums up their shares to obtain an additively shared triple which can be incorrect if a malicious party was dishonest.

In the next step, the **Combine** step, the parties take random linear combinations of the τ components and two randomly sampled values $r, \hat{r} \in \mathbb{F}^\tau$. Thereby, they obtain the two triples where one will be sacrificed later. Following the **Authenticate** step adds MACs to both triples. Because the b is included in both triples, only five values need to be authenticated. Lastly, the correctness of one triple is checked in the **Sacrifice** step. Therefore, all parties first sample a random value s. They then locally compute and open $[\![p]\!] = s \cdot [\![a]\!] - [\![\hat{a}]\!]$. In this context, $[\![.]\!]$ is used to describe a share. With this, they are able to compute the left side of 2 because it is linear. By inserting p and transforming it comes clear that it needs to open up to zero in order for the triples to be correct.

$$s \cdot [\![c]\!] - [\![\hat{c}]\!] - [\![b]\!] \cdot p = [\![s \cdot (c - a \cdot b) + (\hat{a} \cdot b - \hat{c})]\!] \quad (2)$$

With this protocol, the adversary has no chance to cheat. Starting with the **Multiply** stage, any nonzero errors will be detected by the share of the random honest party. This results in an incorrect triple with a high probability. Another approach for the adversary is to guess some bits of a. If all guesses succeed, the triple can be correct and will pass the sacrifice. Thus the adversary learns the bits that were guessed which is called a selective failure attack. However, this is made more difficult by the **Combine** step. To guess a single bit of the final computed share of a, they must guess many bits of the initially generated **a** which is very unlikely to happen. Finally, the **Sacrifice** stage checks the triple with a random value which is unknown when the triples are authenticated. Therefore, it can only pass with a probability of $\frac{1}{|\mathbb{F}|}$ if the triple is incorrect.

Complete preprocessing. By securely generating triples, the main goal of the preprocessing stage was achieved with this protocol. In addition to this, it should also produce random shared values to allow the parties to provide inputs in the online phase. The party simply creates an authenticated additive share of a random value. When later inputting a value, the party broadcasts the difference between the input and the random shared value so that the other parties can adjust their share.

4. Online Phase

The online phase of the MASCOT protocol is quite forward. To share an input x, the party takes a preprocessed random value $[\![r]\!]$ and computes $x - r$. The random value works as a one-time pad and perfectly masks the secret input since it is unknown. After broadcasting the result, all parties compute $[\![r]\!] + (x - r)$ to obtain a share of x.

The multiplication is based on Beaver's circuit randomization technique. Using the multiplication triple (a, b, c) it is straightforward to multiply two secret-shared values $[\![x]\!]$ and $[\![y]\!]$. For this the values $\epsilon = x - a$ and $\rho = y - b$ are computed and opened, where the triple masks the input perfectly as it is uniformly random. Now the sharing of $x \cdot y$ can be computed locally with:

$$[\![z]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$$

To output a share, all previously opened input values are checked. Then the share is opened and verified through the check. If any check fails, the protocol aborts and informs all parties that no value could be computed. Since most computation was moved to the preparation phase, the amount of communication in the online phase is quite small. The only values sent in this phase of the protocol are masked openings for multiplications and outputs. Compared to other implementations, the time for a single multiplication is 200 times faster [1].

5. Conclusion

MASCOT makes faster secure computation of general arithmetic circuits possible. By computing the multiplication triples in the preprocessing phase it allows for a fast online phase without heavy computation. Moreover, through the arithmetic view of oblivious transfer, the COPE protocol succeeds to create sharings of products for the triples. Their generation is based on oblivious transfer extensions where OTs can be realized with fast, symmetric cryptography after a one-time setup. In combination with information-theoretic MACs, it authenticates the triples immediately after creation and when later opening values. By sacrificing another triple the correctness of the multiplication triple is verified. Through the consistency checks and with the sacrifice technique it achieves active security against up to $n-1$ corrupted parties considering a dishonest majority.

In conclusion, the MASCOT protocol improves the SPDZ protocol in the preprocessing phase. Through the reduced computation and communication, it is applicable in the real world by still ensuring active security. It is the first protocol that is making use of oblivious transfer to generate the multiplication triples.

References

[1] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," https://eprint.iacr.org/2016/505.pdf, 2016, [Online; accessed 07-April-2022].

[2] P. K. Madhusudan L, "Information-theoretic macs," https://www.csa.iisc.ac.in/~arpita/Cryptography15/CT4.pdf, 2015, [Online; accessed 07-April-2022].

[3] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 479–488, 1996.

[4] N. Gilboa, "Two party rsa key generation," *Advances in Cryptology - CRYPTO*, pp. 116–129, 1999.

[5] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," *Advances in Cryptology - CRYPTO*, pp. 145–161, 2003.

Shortest Path Awareness in Delay-Based Routing

Mia Heinz, Christoph Schwarzenberg*, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mia.heinz@tum.de, schwarzenberg@net.in.tum.de, wiedner@net.in.tum.de

***Abstract*—The back-pressure routing (BP) algorithm is proveably throughput optimal which makes it a very promising algorithm, but it struggles with high end-to-end delay. In this paper, we will compare existing approaches on delay-based routing, shortest-path-aided back-pressure routing and possible combinations of both to determine how they decrease the end-to-end delay while maintaining throughput optimality.**

***Index Terms*—back-pressure, shortest path, delay metric, end-to-end delay**

1. Introduction

Efficient and fast routing is becoming increasingly important especially in real-life applications, like streaming where a large end-to-end delay is very noticeable.

The BP algorithm promises throughput optimality which means that it guarantees system stability in a network. The network is stable meaning that queue occupancy does not increase endlessly.
Under a high traffic load the algorithm works comparably well as it exhausts all possible paths and therefore distributes the traffic over the whole network instead of concentrating it on one or few paths.
If the traffic is lighter, however, this becomes a problem. With a low traffic load the algorithm chooses randomly between all possible paths, therefore sending data over unnecessary long paths or even loops causing high end-to-end delay2.

First, the paper describes the BP algorithm in Section 2. Section 3 gives an overview of existing optimization methods and in Section 4, we focus on shortest-path-awareness and delay-based algorithms.

2. Background

The BP routing algorithm was first proposed by Tassiulas and Ephremides in [1]. It was originally developed for wireless multihop radio networks but can easily be transferred to wired multihop networks.

The algorithm works with congestion gradients in queueing networks. Each node has a queue for each destination in the network. A flow is a sequence of packets belonging to one activity. A link weight is calculated by the difference of queue length for a flow and the neighbor's queue length for the same flow. The biggest weight signifies the least congested path. Then in each time slot, the scheduling decision is made to maximize the sum of the weights for activated links.

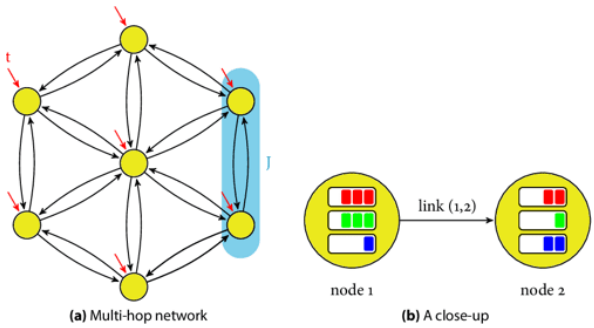


Figure 1: General principle of the BP algorithm [2]

Figure 1 shows an example illustrating a multihop queueing network. For the shown link $(1, 2)$ in this example the BP algorithm would choose the green flow, as the difference in queue length is the highest.

Another way to understand the principle is imagining the links of a network as pipes and the data as water. The water is put into the pipe network in one place and can only escape at its destination. If it is only a little bit of water it will randomly distribute over the pipes. If more water is introduced to the network, the water pressure pushes the water out at the destination. With the BP algorithm pressure builds up leading the data to its destination the same way water pressure builds up and pushes the water out of the pipes at the destination.

The BP algorithm is provably throughput optimal and stable. But challenges are the high end-to-end delays, especially in low data load traffic because it always exploits all possible paths.
According to Hai et al. in [3], the high end-to-end delay is caused by three main factors:

1) The *initial delay* describes the initial startup time it takes to build up pressure for the BP algorithm to work.
2) The *random walk delay* is the delay that is created when two links have the same weight, so a random choice is made which can lead to looping or unnecessary long paths.
3) The *last packet delay* describes a delay of the last packet of a flow. If no packets to the same destination are following the last packet of a flow, the queue remains lightly filled and the packet may starve for an undefined amount of time.

The following section analyses approaches to reduce the end-to-end delay.

doi: 10.2313/NET-2022-11-1_07

3. Related Work

Since the first introduction of the BP algorithm there have been many attempts to improve it.

The first group of optimization methods is based on queue structure and management.

One approach is the clustering of nodes, which was explored by Ying et al. in [4]. The idea is that nodes are combined in clusters and if the source and destination are already in the same cluster, the standard BP algorithm is applied. If they are in different clusters, the BP algorithm is used to reach one gateway of the cluster containing the destination. This not only reduces the end-to-end delay but also the memory complexity, as fewer queues have to be maintained. The main problem of this algorithm is the question of how to cluster the nodes.

In [5] Alresaini et al. introduce a BP algorithm with adaptive redundancy (BWAR). If a node has a queue backlog below a certain threshold, it duplicates the packets it is sending in another buffer. Then when the queue is empty those copies are sent. By creating several copies of one packet, more pressure is built up and the chance that one arrives at the destination is higher. When one of them arrives at the destination, all other copies have to be deleted. Deleting all copies is a problem which can be solved with timeouts.

Another approach is combining the BP algorithm with data science and machine learning. In [6] Huang et al. propose a predictive BP algorithm that predicts and preserves the arriving packets based on a lookahead-window.

Furthermore, approaches that reduce the delay by decreasing the path length or the number of hops, like avoiding or reducing loops [7] or including shortest-path-awareness [8], will be considered in detail in Section 4.1.

Delay-based algorithms, that use metrics other than the queue length to make the routing decisions, can also be used to reduce the delay, which will further be explained in Section 4.2.

4. Approaches

In the following we will have a closer look at the approaches using shortest-path-awareness and delay-based routing.

4.1. Shortest-Path-Awareness

In [8], Ying et al. combine the traditional BP algorithm with shortest-path-awareness based routing which decreases end-to-end delay while maintaining throughput optimality.

The end-to-end delay is dependent on the hop count of packet i.e., the path length and the time it spends in the queues. Using an algorithm like Dijkstra or Bellman-Ford the shortest path to each destination can be calculated for every node.

If the routing algorithm would now always use the shortest path, the path becomes congested very fast and this leads to a very high queue delay as the packets are buffering and waiting in the queues.

Therefore, in the proposed algorithm not only the shortest path is used but all paths with a specified maximum path length. This hop constraint gets adjusted as the data rate in the network rises.

First, Ying et al. propose an algorithm for a specified hop constraint h, which uses the BP algorithm but only exploits paths with a length $\leq h$. So every node knows the minimum hops for every destination and every flow has a hop constraint h. Then the path (a, b) is only a valid option if node b is less than $h - 1$ hops away from the destination. From all possible hops, the one with the least congestion is picked via the BP algorithm.

In a network we usually do not have a set hop constraint but the maximum path length is dependent on the data rate and h has to be chosen dynamically. They introduce a variable K which is the price for taking a longer path, so a larger K minimizes the average number of hops but leads to a larger queue delay as not that many paths can be used. A smaller K means that longer and therefore more paths can be used, better distributing the flows and leading to a smaller queue delay.

Using this K, the optimal h is calculated dynamically to minimize the tradeoff between hops and queue delay.

The combination of the BP algorithm and the shortest-path-aided BP algorithm significantly reduces the initial delay as in the beginning no pressure is needed to guide the data to its destination and just the shortest path is used. The random walk delay is also decreased because if links have the same weight, now the shorter path is chosen and loops are prevented by limiting the hop constraint to the number of nodes.

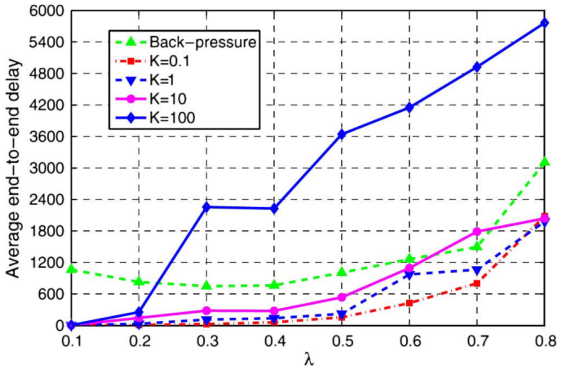


Figure 2: Simulation results comparing end-to-end delay in traditional BP algorithm and shortest-path-aided BP algorithm with different values for K using OMNeT++ with λ arrival rate of flows in packets/time slot [8]

As we can see in Figure 2 for a low data rate the delay of the BP algorithm first decreases before increasing again with higher data loads. This is due to the initial packet delay. In the shortest-path-aided BP algorithm the end-to-end delay is significantly reduced compared to the traditional BP algorithm, as especially under a low data rate excessively long paths are not explored. But also for higher data rates, there is an improvement as the random walk delay is reduced and loops are prevented. However, the problem of the last packet delay is not solved as starvation is still possible.

But it all depends on the value of K. If K chosen is too large, the algorithm always chooses the shorter path independent from the queue backlogs, because the penalty of choosing a longer path is too high. Hence, the delay

doi: 10.2313/NET-2022-11-1_07

caused by buffering in the queues is not considered. If K is too small, however, unnecessary long paths are taken even if the shorter path would have been faster. The optimal value for K is specific to the network and the problem of how to choose it is still open.

In [7], Rai et al. developed a loop-free BP (LFBP) algorithm. By giving the links in the network a direction a directed acyclic graph (DAG) is created. Then in this DAG the BP algorithm is applied. If it now overloads at some point in the network, the directions of the links pointing from the non-overloaded nodes to the overloaded nodes are reversed which creates a new DAG. This procedure prevents congestion in the network by routing packets away from overflowing areas and towards less busy areas of the network.

Another hop minimizing algorithm is presented in [9] by Bui et al. The Enhanced Backpressure (EBP) algorithm uses shortest path heuristics to first use short paths and only add longer ones if the links are overloaded.
This algorithm also uses shadow queues and only maintains queues for the direct neighbors, which reduces the memory complexity as well as the delay.

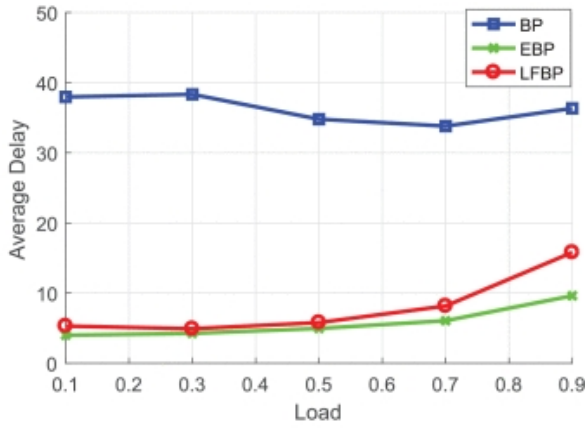


Figure 3: Simulation results comparing end-to-end delay in traditional BP algorithm, EBP and LFBP [7]

As we can see in Figure 3, both the EBP and the LFBP have a far smaller delay than the traditional BP algorithm with the EBF showing slightly better results. One can especially see that both algorithms do not suffer from the initial packet delay.

4.2. Delay-Based Algorithms

In [3], Hai et al. introduce a delay-based optimization method, which combines the queue length with the packet delays to calculate link weight. They introduce a metric called the sojourn time backlog (STB) and an STB-based BP algorithm (STBP). Instead of the length of the queue the sum of sojourn time, the time passed since a packet arrived in the network, of all packets in the queue is used.

If every packet just has the weight 1, like in the traditional BP algorithm, there cannot be a prioritization of packets, which already have a high delay. Giving the packets different weights by assigning the STB as the link weight, leads to more pressure on packets with a higher delay, preventing starvation and unnecessarily long paths, therefore reducing the average end-to-end delay. Hai et al. introduce an implementation of this algorithm using First-In-First-Out (FIFO) queues and virtual queue management.

As synchronization in networks is very hard to achieve, another option Hai et al. propose is taking the hop count as the delay instead of the actual time. Using this hop approach the delay of the packets is not increasing while buffering, therefore packets can be stuck in queues for a very long time and starve without their priority increasing. It still reduces the average number of hops and the end-to-end delay because if a packet was already transmitted over many hops, it is then prioritized and the remaining number of hops is therefore on average smaller.

Both versions decrease the initial delay and the random walk delay, but the last packet delay is only decreased by the STBP using the actual time and not the hops because the starvation problem is not fixed.
Considering flow dynamics the last packet delay is especially important as all packets of a flow have to be transmitted. So in this context the STBP algorithm using the actual time has a significant advantage over the hop count based version.

In [10], McKeown et al. introduce another weight metric. It uses the sojourn time only of the head-of-line (HOL), the first packet in the queue. The so-called oldest cell first (OCF) algorithm, which uses HOL delay as a metric to calculate the link weights, achieves less delay compared to the traditional BP algorithm.
In the BP algorithm queues with a short length can be starved if the length remains small and other queues receive new packets regularly, therefore being prioritized over the shorter queues. This problem is solved with the OCF algorithm as the waiting time of the HOL increases in each time slot. This way the HOL delay gets bigger until it is eventually served.
The OCF algorithm also reduces the last packet delay as it prevents starvation of shorter queues by only considering the first packet.
Another HOL delay-based algorithm is introduced by Ji et al. in [11].

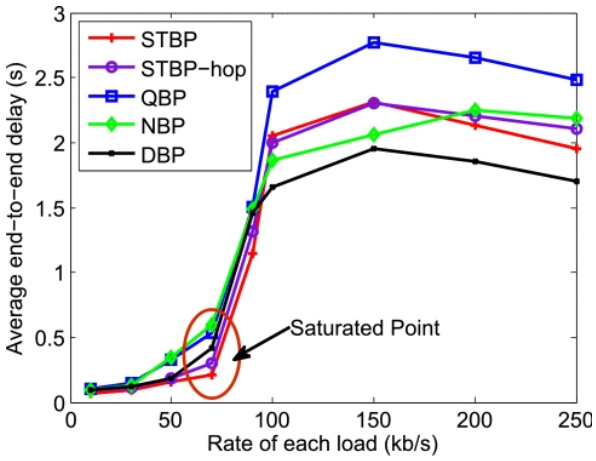


Figure 4: Simulation results comparing end-to-end delay in traditional queue-based BP (QBP), the STBP, the hop based STPB (STPB-hop) and an HOL delay based algorithm (DBP) using NS-2 network simulator [3]

In Figure 4 the network saturation point at about

doi: 10.2313/NET-2022-11-1_07

70kb/s is visible. After this point, the delay of all algorithms increases sharply. According to Hai et al. after the saturated point the network is no longer able to stabilize the data rate and operates in an overload.
When looking at the range where the network is still stable, we can clearly see the biggest improvement in end-to-end delay is achieved with the STBP but the hop-based STBP and the HOL delay-based algorithm are still better than the traditional BP algorithm.

When comparing the STBP, the HOL delay-based based algorithm and the traditional BP algorithm it becomes obvious that the STBP, which is a combination of the other two has the best performance. The end-to-end delay gets higher the longer packets are buffering in the queues.
The HOL algorithm ignores very long queues if the first packet only has a small delay, as the algorithm does not consider queue length. As the delay increases constantly, these longer queues are not starved. The longer a packet is not served, the higher the delay gets and the more likely it will be served in the next time slot. It can however lead to a slightly bigger delay for the waiting packets.
The queue-length-based BP on the other hand ignores short queues even if the packets in it already have been waiting for a long time. The STBP prioritizes long queues and highly delayed packets combining the advantages of both algorithms.

Another approach is the LIFO-backpressure which is explored by Huang et al. in [12]. They show that by simply combining FIFO and Last-In-First-Out (LIFO) queues a significant improvement in delay can be achieved as the packets with the highest delay can be served first regardless of if they are at the head or tail of the queue.

4.3. Comparison and Combination

The algorithm reducing the path length and delay-based algorithms focus on different aspects.
Both algorithms still have unsolved problems for example how to choose K in the shortest-path-aided BP algorithm or how to achieve synchronization but both approaches already accomplish a significant improvement compared to the traditional BP algorithm.

By reducing the path lengths a packet is directed to its destination faster but packets can still starve and especially with a high data load almost as many paths are used as in the traditional BP algorithm.

In delay-based algorithms we create a prioritization of packets that already have a high delay but the path is still chosen randomly and loops can still occur.

So naturally one could try combining both algorithms to decrease the end-to-end delay even more and optimize the BP algorithm further. One idea would be using the joint algorithm from [8] but instead of the queue length the STB is used as the metric to make the routing decisions. By combining the algorithms it would be possible to profit from the shortest-path-aided BP algorithm reducing the initial packet delay and the STBP reducing the last packet delay and both algorithms reducing the random walk delay. This would decrease the overall delay even more. However, when combining the two algorithms one also has to consider the challenges of both algorithms.

Especially in flow-based routing we cannot only look at per packet delay but have to consider the overall delay until all packets of the flow have arrived. Therefore the last packet delay plays an important role, as all packets of a flow have to arrive.

5. Conclusion

We analyzed and compared the shortest path and delay-based approach. Both reduce the end-to-end delay and have their specific advantages and challenges. For shortest path based approaches the biggest improvement is the reduction of the hops in low data load and the biggest challenge is the tradeoff between minimizing the path lengths and the delay from buffering in the queues. But if an optimal parameter K is chosen for this, the end-to-end delay is also reduced for higher data loads, e.g. by avoiding loops.

Delay-based algorithms reduce the end-to-end delay by prioritizing already heavily delayed packets and bringing those to their destination first. The challenge with this is how to measure the delay as synchronization in networks is very hard to achieve and also while they are prioritized, it is still possible that the packets are transmitted in loops or overly long paths.

All in all a combination might benefit from both algorithms' strengths and lead to an even better optimization but we have to consider that we also have to deal with both algorithms' challenges.

References

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," in *29th IEEE Conference on Decision and Control*, 1990, pp. 2130–2132 vol.4.

[2] J. Kampen, "Route guidance and signal control based on the back-pressure algorithm," 2015.

[3] L. Hai, Q. Gao, J. Wang, H. Zhuang, and P. Wang, "Delay-optimal back-pressure routing algorithm for multihop wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 2617–2630, 2018.

[4] L. Ying, R. Srikant, and D. Towsley, "Cluster-based back-pressure routing algorithm," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 484–492.

[5] M. Alresaini, M. Sathiamoorthy, B. Krishnamachari, and M. J. Neely, "Backpressure with adaptive redundancy (bwar)," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 2300–2308.

[6] L. Huang, S. Zhang, M. Chen, and X. Liu, "When backpressure meets predictive scheduling," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2237–2250, 2016.

[7] A. Rai, C.-p. Li, G. Paschos, and E. Modiano, "Loop-free back-pressure routing using link-reversal algorithms," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2988–3002, 2017.

[8] L. Ying, S. Shakkottai, A. Reddy, and S. Liu, "On combining shortest-path and back-pressure routing over multihop wireless networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 841–854, 2011.

[9] L. Bui, R. Srikant, and A. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," in *IEEE INFOCOM 2009*, 2009, pp. 2936–2940.

[10] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

doi: 10.2313/NET-2022-11-1_07

[11] B. Ji, C. Joo, and N. B. Shroff, "Delay-based back-pressure scheduling in multihop wireless networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1539–1552, 2013.

[12] L. Huang, S. Moeller, M. J. Neely, and B. Krishnamachari, "Lifo-backpressure achieves near optimal utility-delay tradeoff," in *2011 International Symposium of Modeling and Optimization of Mobile, Ad Hoc, and Wireless Networks*, 2011, pp. 70–77.

doi: 10.2313/NET-2022-11-1_07

Digital Twins of Computer Networks

Jacqueline Kroyer, Kilian Holzinger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: kroyerjackie@gmail.com, holzinger@net.in.tum.de

***Abstract*—Digital Twin Technology such as Digital Twins (DTs) and Digital Twin Networks (DTNs) are part of an emerging trend in the operation of computer networks. However, in order to apply the appropriate digital twin technology, their functionality, core goals and technical requirements have to match the field of application. By examining digital twin technology and providing a structured overview of the relevant parameters around terminology, goals and requirements of DTs and DTNs, this seminar paper contributes to existing research in the field of DTs. A widely pursued goal is to leverage all available machine data by applying digital twin technology.**

***Index Terms*—Digital Twin, Digital Twin Technology, Digital Twin Network, Computer Network, P2V Communication**

1. Introduction

The idea of leveraging state of the art computer network technology in the field of machine to human communication is highly relevant as it bears the potential to revolutionize many industries as well as our every day lives. Digitization of the industry is continuously progressing and its demands are increasing. The process is referred to as Industry 4.0 and requires a tremendous amount of machine data. The development of digital twin technology can e.g. be harnessed for more efficient and sustainable cities. Applying digital twin technology goes beyond smart cities: manufacturing, aviation, healthcare or transportation systems are just a few examples.

This seminar paper researches and examines digital twins of computer networks. It focuses on the question which technical requirements have to be met for respective technological core goals and their implementation in the form of fields of applications.

With the increased amount of personal data and especially the increased amount of available machine data, the statistical relevance increases as well as the reliability of the models used for the virtual twin and all elements related to the digital representation of the physical object.

Within this seminar paper, after outlining the theoretical background of digital twin technology by defining the most relevant terms the functionality of digital twins and digital twin networks will be examined. Building on technical requirements of DTs and DTNs, core goals will be derived and illustrated in which fields of application DTs and DTNs are most relevant. In the following theoretical foundations such as relevant definitions around digital twins, their functionality as well as technical requirements and core goals will be illustrated.

2. Theoretical Background of Digital Twins of Computer Networks

In order to provide the reader with the relevant, theoretical underlying concepts, the following section defines DTs and DTNs, their functionality as well as the different types of DTs.

2.1. Digital Twins and Digital Twin Networks

" [...] DT is an intelligent and constantly evolving system, which monitors, controls, and optimizes the physical object through its life-cycle." [1]. This system consists of one or more physical and virtual interconnected objects. Figure 1 shows the typical concept of a DT. The arrows connecting the physical with the virtual twin represents the transmission of information and decision-making assistance. It also indicates the direction of the data flow.

Three aspects related to DTs are especially relevant. Firstly, the type of connection between the physical object and the physical object is a one-to-one connection. Secondly, the concept is comprehensive as it contains and relies mostly on these two objects. Thirdly, the static form of a DT is a perfect virtual representation of the physical object and therefore can be a mere simulation of the status-quo.

The formal definition of DT by Michael Grieves emphasizes the three elements of a physical object in the physical space, a virtual object in the virtual space and the data link between the two objects and spaces [2]. From an industry perspective, this definition is tightly knit to concepts in product engineering [2]. Examples for physical objects are machines, humans or human-related things like in a smart city. Types of physical objects are complex physical systems, machines, robots or industrial processes [1]. The concept of a virtual twin in general can also be described as a virtual representation of the physical object. It is a virtual 3D model [3]. Consequently, it is arranged and conceptualized as similarly as possible to the corresponding physical object. A physical object and its virtual twin can be developed together and are connected via a bi-directional data flow which also marks a shift from a static to a dynamic scenario.

doi: 10.2313/NET-2022-11-1_08

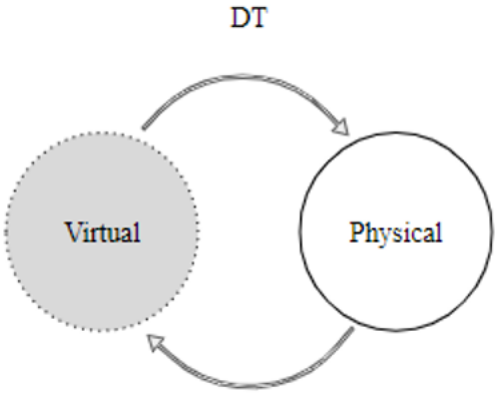


Figure 1: Concept of a DT; information transmission and decision-making assistance; own illustration based on [1];

DTs can be categorized in three types, depending on perspective, focus and type of connection.

- 1st DT type: The first type covers the exact virtual representation of a physical object. It has no automatic connection to the physical twin and is characterized by no data exchange between physical object and virtual object.
- 2nd DT type: The second type possesses a unidirectional connection to the virtual twin and is therefore able to illustrate an evolution of the physical object.
- 3rd DT type: The third and last type is the most enhanced category of DTs and includes services and the respective connection among the objects. Thus, it enables mirroring the current state of the physical object because it continually adapts and predicts future states.

As a second crucial term within the field of DT technology, DTNs will be explained. Within the course of this paper, DTNs can be defined as multiple one-to-one DTs, building a many-to-many mapping network [1].

This means the physical object is connected to other physical twins as well as to the virtual twin and its virtual twins which enables real-time information interaction. By applying key technologies such as communications, accurate DT modeling, physical data processing, cloud and edge computing, a DTN can capture dynamic interactions and evolutions of multiple physical as well as virtual objects [1].

Figure 2 illustrates the typical concept of a DTN. The ways of communication are a physical-object-to-physical-object connection (P2P), a virtual-twin-to-virtual-twin connection (V2V) as well as the typical physical-object-to-virtual-object connection (P2V). For P2V connections shared intelligence and cooperation is applied. For a P2V connection, information is transmitted. It also serves decision-making assistance. Three aspects about DTNs are important to emphasize. Firstly, the type of connection between the physical objects and the virtual twins is a many-to-many connection. Secondly, the physical objects and virtual twins aren't isolated. Consequently, it is a cooperation approach. Thirdly, when creating and conceptualizing a DTN, the physical objects and the virtual twins are put in relation and co-developed. Therefore, the physical objects and the virtual twins evolve together. This concept is called co-evolution.

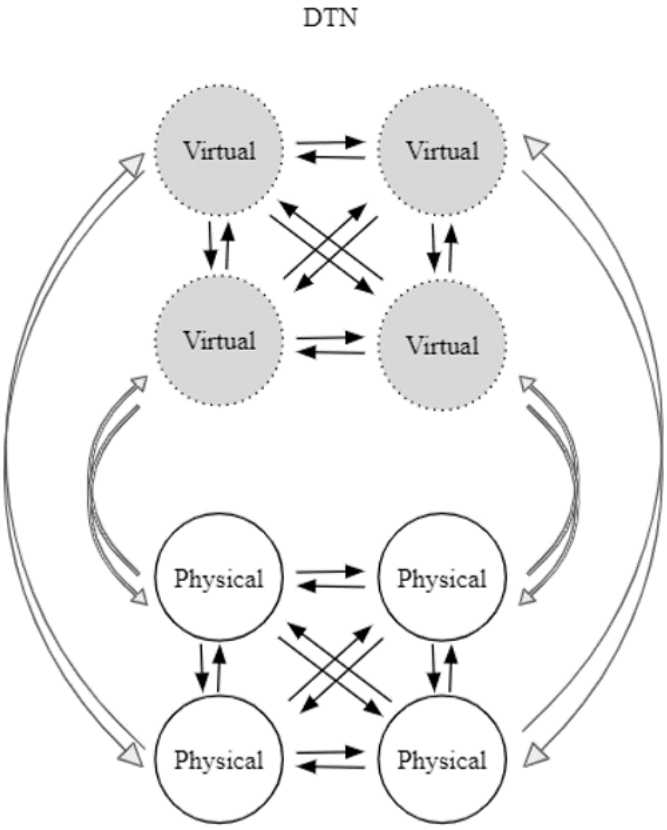


Figure 2: Concept of a Digital Twin Network: information transmission and decision-making assistance (P2V); shared Intelligence and Cooperation (P2P, V2V); own illustration based on [1];

When comparing DTs and DTNs one quickly notices a shift from a static to a dynamic scenario [4]. This means, time independent, three dimensional objects turn into actionable CAD objects. Thereby changes can be simulated and behavior can be predicted more accurately and precisely over time.

So called cyber-physical systems (CPS) are often mentioned in the context of DT technologies. It is important to terminologically differentiate a DT and a DTN from a CPS. A CPS is defined to be a "physical and engineered system whose operations are monitored, coordinated, controlled and integrated by a computing and communication core" [5]. A DT, DTN or CPS differ in application scenarios. DTs and DTNs are often used in an industrial context whereas a CPS is more often applied in embedded systems due to physical properties like sensors or actuators. Furthermore, it is important to highlight that

doi: 10.2313/NET-2022-11-1_08

DTs and DTNs are model-based systems that are either applicable for a single object (DT) or a group of objects (DTN).

2.2. Functionality

When looking at the functionality of DTs and DTNs one has to understand the underlying key technologies. As briefly mentioned previously, these are namely communications, physical data processing, DT modelling, cloud computing as well as edge computing [1]. In the following they will be explianed in more depth.

Communications are focused on the type, direction and content of exchange of information. This can be P2P, P2V or V2V communications:

- Physical object to virtual twin: P2V communications describe the exchange of data of a physical object with a virtual twin through the means of wireless communication technologies. This exchange and transmission of data can be conducted in real-time. In addition to that the virtual twin is able to receive and apply feedback provided by the physical object. Suitable networks are LoRa or 5G/6G cellular communication networks. Accurate mapping and real-time feedback are desired goals.
- Physical object to physical object: P2P communications refer to exchange of information between two physical objects. By connecting with IoT gateways or Wi-fi access points, RFIDs, actuators or controllers of the respective physical devices the connection between two physical objects can be established. The network connection itself is supported by several communication protocols such as Wireless Personal Area Network or Low Power Wide Area Networks.
- Virtual twin to virtual twin: V2V communications reflect ongoing communications between two physical objects. As virtual twins can be seen as virtual replicas they mirror the exchange of data between physical objects. Hence, V2V transmission of data occurs between DT model entities. The virtual level of communication relies on computing capabilities in order to mirror the data transmission behavior in the respective DT model.

Besides communications, it is important that the raw data that derives from multiple sources is appropriately handled as it is typically still full of noise. This is specifically relevant in order to make effective use of the data as the amount of raw data is likely to increase with the increasing number of sensors and data sources.

Furthermore, it is important to clarify the applied DT model and framework early in the process. Although there are several options, the state-of-the-art framework would be Tao et al. 's who suggest a DT five-dimensional model taking the physical part, virtual part, data connection and service modeling into consideration [6].

Cloud Computing is a key technology in the context of DTs as it enables large-scale computing. Thereby, it is a crucial element to allow sharing processes on demand and also facilitating the option to use the services anytime and anywhere.

As a last key technology, edge computing uses new computing models for several operations such as analysing or storing data. It is a solution for privacy protection, reduced latency and also reduces power and costs as well as makes the system more reliable.

It takes all the mentioned technologies to efficiently handle processing data and ensure well-functioning DT or DTN communication.

3. Digital Twins of Computer Networks

In the context of DTs and DTNs the respective core goals as well as the technical requirements to meet these goals are important building blocks to properly apply DT technology. Examples for fields of applications will be derived from technical requirements and core goals.

3.1. Core Goals and Technical Requirements

All of the addressed key technologies of DTs and DTNs, including the three types of communications (P2P, P2V, V2V), physical data processing, digital twin modeling, cloud computing as well as edge computing possess certain technical requirements that have to be met. This specifically applies to the types of communications of DTNs.

In the context of this seminar paper the most important technical requirements are addressed. Namely they are:

- Low latency for real-time feedback: In most situations and scenarios real-time feedback and interactions are necessary. The P2V connection determines the required latency needed for the specific scenario. The higher the time sensitivity for the physical object as well the data connection is the lower the latency has to be. Medical use cases require ultra-low latency.
- High transmission reliability for accurate modeling: Accurate data needs to be exchanged in a reliable and immediate way in order to implement and make use of dynamic high-fidelity modeling approaches. Depending on the use case either medium transmission reliability is sufficient. In certain use cases like medical scenarios, it has to be especially high.
- Secure data transmission for higher data privacy and security: At the level of data exchange, mostly between the physical object and the virtual twin, the security of data and its privacy are of high priority. Typical protocols and encryption methods

doi: 10.2313/NET-2022-11-1_08

of the field of IT security need to be applied in order to guarantee safe and private data exchange.

- Higher network bandwidth and capacity increase than rise of demand of interconnected elements in the network for reliability and availability: As sensors deliver a much greater amount of data the total amount available increases which is helpful in terms of prediction accuracy of the relevant models. Due to the sheer amount of data, it remains challenging to process the data. Therefore the bandwidth as well as the capacity of the network has to increase faster than the demand of the risen number of interconnected objects.

In total, all of these requirements, namely, low latency, high transmission reliability, secure and private data exchange as well as a fast enough increase of network demand and bandwidth are crucial but also strongly dependent on the use case and therefore the industry in which the DT or DTN is applied.

The corresponding core goals are the provision of real-time feedback, accurate modeling, privacy and security as well as reliability or respectively availability.

3.2. Comparison of DT and DTNs

When comparing DTs and DTNs, several parameters are relevant and decisive to consider: The modelling approach, interaction, collaboration with other models, efficiency, accuracy, components and mapping relationship. Table 1 shows how DTs and DTNs differ or resemble.

Both categories still aim at accurate modeling, real-time feedback as well as the higher level goals of security, privacy and availability.

	Comparing DTs and DTNs	
	DT	***DTN***
Modeling Approach	Single, independent object	Group of objects with complex internal interactions
Interaction w. other models	Not applicable; no interaction with other models	Cooperation approach; processing results shared among collaborative DTs
Accuracy	Mirroring approach; Virtual twin as accurate digital replica	Higher; Physical and virtual twin evolve together
Components	Physical object and virtual twin; P2V data exchange	Physical objects and virtual twins; P2V, P2P and V2V data exchange
Mapping Relationship	One-to-one method	Many-to-many method

Table 1: Comparison between the concept of DTs and the concept of DTNs; own illustration slightly based on [1]

The direct comparison of DTs and DTNs emphasizes how the two concepts tend to differ. Depending on the field of application, the specific industry or even just the use case, one of the two concepts might fit better than the other. Furthermore, certain aspects like the exact definition of the technical requirements will have to be adjusted.

3.3. Fields of Application

The use cases for DT technology are broad and have strong advocates in the industry. Microsoft's HoloLens for example can be applied to view the three dimensional, virtual model of a physical smart factory. Moreover it can then be used to control some of the production's functions remotely. Batty also names industries, cities and communities as some of the most relevant application fields [2].

The most relevant industries and use cases are manufacturing, aviation, healthcare, 6G networks, intelligent transportation systems as well as urban intelligence [1]. In each of the examples DT technologies bear vast potential to add more efficiency to the related business processes, increase the control over what processes are actually conducted in real-time and enhance monitoring capabilities. This means that certain insights, like the current energy consumption, can be displayed or intelligent control mechanisms of production processes in manufacturing can be leveraged.

In summary, the virtual twins can be generally seen as human-centric user interfaces for data exchange and specifically for the transmission of digital information from the physical object [3].

4. Conclusion

DT technology is a field of high relevance for manifold technologies evolving based on DTs or DTNs. It is crucial to understand the differences in the technologies in order to properly apply, implement and efficiently use it.

Core goals like reliability, latency, capacity, connectivity and efficiency require different values depending on the type of communication: P2V, P2P or P2V. It is also decisive to depend on the design of the DT technology on the type of use case it is applied to. The most relevant fields of application are around business and production contexts like manufacturing or key use cases in medicine and health care.

As DTs and DTNs bear tremendous potential for our economy and foster technological progress and innovation, further research has to be conducted to meet the technology's potential.

Acknowledgment

This work was not supported by any grant. I want to express my gratitude to my advisor Kilian Holzinger who supported me throughout the writing process from the beginning. Because of his idea for this highly relevant and interesting topic I got the chance to explore it further.

References

[1] Y. Wu, K. Zhang, Y. Zhang (2021) Digital Twin Network: a Survey, in: IEEE Internet of Things Journal, pp. 1-17.

[2] M. Batty (2018) Digital Twins, in: Environment and Planning B: Urban Analytics and City Science, pp. 817-820.

[3] M. Kritzler, M. Funk, F. Michahelles, W. Rohde (2017), The Virtual Twin: Controlling Smart Factories Using a Spatially Correct Augmented Reality Representation, in: IoT '17: Proceedings of the Seventh International Conference on the Internet of Things, pp. 1-2.

[4] M. Grieves, J. Vickers (2017) Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior, in: Complex Systems, pp. 85–113.

[5] R. R. Rajkumar, I. Lee, L. Sha, J. A. Stankovic (2010) Cyber-physical Systems: The Next Computing Revolution, in: Design Automation Conference.

[6] F. Tao, H. Zhang, A. Liu, A. Y. C. Nee (2019) Digital Twin in Industry: State-of-the-art, in: IEEE Transactions on Industrial Informatics, pp. 2405–2415.

doi: 10.2313/NET-2022-11-1_08

Secure Data Marketplaces

Daniel Petri Rocha, B.Sc., Dr. Holger Kinkelin*, Filip Rezabek, M. Sc.*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: daniel.petri@tum.de, kinkelin@net.in.tum.de, frezabek@net.in.tum.de

***Abstract*—Big data powers the growing data economy. But critical data sets needed for research and development remain isolated. Those selling such data have no means of preventing copies from being created. This paper summarizes the building blocks required to construct a secure data market, where privacy and control are inherently built into the system despite large-scale information access remaining possible. Mechanisms for granting access to data as desired by the owner are described, enabling data to be leased without exposing it. Secured data processing techniques and blockchain technologies are suitable for assembling privacy-preserving data marketplaces.**

***Index Terms*—data market, data silo, blockchain**

1. Introduction

Data is valuable — the European Union's data economy alone is forecast at 550 billion Euros by 2025 [1]. The analysis of vast data collections fuels technologies that aid in the accurate and deep understanding of areas of societal importance, yielding, for instance, advancements in scientific research. Big data sets help uncover treatments for severe illnesses by studying the human genome [1]; healthcare centers can improve patient care with shared information [2]. However, trading such data often does not happen in practice, given that a separate entity with a copy could choose to redistribute it indiscriminately.

Therefore, enterprise information holders keep a monopoly on highly demanded data even though they know its value [3]. These data silos are a financial loss and liability source since a breach would leak business secrets or personally identifiable information without usage restrictions, artificially imposing a cap on the data's potential [3]. Instead, incentives should make organizations provide data to others in a discoverable and integrable manner [4].

A way to accomplish this without exposing sensitive information is by selling data as a good or service on secure data marketplaces. They offer the tools needed to create an additional revenue stream for data holders while ensuring the owner's data privacy and control [2]. Consumers use the market to locate and access the data they need without trusting a central authority. They can then work with it perpetually, only for a period or a limited number of times [3]. Interactions against the bought data occur with programs, e.g., running queries or machine learning algorithms to get a trained model back. Unlike downloading a file or tapping a stream, data is employed without risking it being cloned as it does not leave the holder's premises.

This paper is structured as follows: Section 2 describes where and how such technology is being applied or envisioned to be. Section 3 explains the desirable properties of data markets in conjunction with their motivation. Section 4 addresses what a data market consists of. Section 5 provides approaches to designing, implementing, and maintaining data marketplaces with the identified parts and properties. Section 6 concludes this proceeding.

2. Application Scenarios

Data marketplaces available today meet specific information demands. The subsections below analyze prominent data market applications and where to find them.

2.1. Ocean Market

The Ocean Protocol is a project attempting to supply an interface to simplify setting up data markets. Their open-source protocol provides the necessary infrastructure to give and withdraw paid data access. A transport company could increase revenue by using Ocean to deploy a data market for annotated dashcam footage they currently silo, which may be of interest for the computer vision models of the automakers engineering self-driving vehicles.

An exemplary Ocean-powered market is the Ocean Market[1], on which users pay with their crypto wallet. In return, they redeem a token for that asset, which can be considered a license of the original data set. Depending on the fine-grained permissions set by the data owner, this license is the ticket to data services such as downloading a copy or using it as input for an algorithm without revealing the underlying data. The provider can approve or deny programs to avoid privacy infringements. Ocean itself does not store any data: ownership corresponds to minting a non-fungible token on the Ethereum blockchain pointing to an external resource [3].

However, a decentralized identifier for the resource is stored on-chain, together with a separate document offering a metadata description to make it easily discoverable. For precise data control access, required credentials may be embedded in the metadata store, representing an additional type of identification besides token ownership. Using a role-based access control server whose implementation Ocean provides, capabilities around service consumption can be restricted.

1. https://market.oceanprotocol.com/

doi: 10.2313/NET-2022-11-1_09

Besides self-hosting a secure market, opening up siloed databases, or refining public data sets by making them effortlessly integrable and discoverable, an additional income stream can come from staking on data. Since Ocean Protocol data sets are tokenized and have value, prices can automatically be determined by an automated market maker — effectively transforming the data set into a cryptocurrency. The market maker dictates how expensive a token should be given its availability in a liquidity pool. The pool's liquidity is defined by the number of data tokens and cryptocurrencies such as Ethereum or Ocean (the organization's coin) it holds. An asset's cost increases as it is purchased and used, paying dividends to those providing liquidity. It decreases when data tokens are sold since the automated market maker derives the price from the assumption that the ratio of data tokens to, e.g., Ocean should stay constant at 50:50. Thus, a monetary incentive exists to curate data, which provisions liquidity [3]. An engineer, satisfied with the better rate of red traffic lights detected in their model after using a data set, may choose to stake it for profit. That, in turn, signals the market that the data set's quality is high.

2.2. Kara

Kara[2] is a secure market for medical data whose goal is to improve research and outcomes in medicine by recognizing that valuable health-related databases are siloed and offering a user-centric solution that lets patients share data themselves. The Oasis blockchain [5] fuels it.

After a doctor's visit, e.g., to perform a back of the eye scan, they let patients upload that image to a medical data cloud in exchange for cryptocurrency alongside a policy of use. An example guideline the patient provides could be to revoke their scan's use for commercial purposes or only grant research access for a certain number of years. They remain the data owner at all times instead of an intermediary company they may not trust.

Collaboration is fostered among doctors and scientists as the data units are inherently sharable due to a privacy-preserving architecture, ensuring the unencrypted scan can never be seen. Nonetheless, surveying the information remains possible, with applications including performing statistical analysis for genomics research and training machine learning models against the data set.

3. Properties of Data Markets

Transactions in a data marketplace occur between a data supplier, which owns a unit of information they are willing to sell, rent, or barter, and a consumer ready to enter the trade to obtain access [6]. As rational participants, a guarantee that a subset of the characteristics described below is enforced may interest both parties and the marketplace network to create a growing self-sustaining environment.

Data as (crypto-)currency A *currency* is a medium that facilitates trade. As with money, data is an asset belonging to an individual and can therefore be classified as a currency [2]. Personal data is part of how access to some online services remains free, being given up as payment. Data markets could enable users of these platforms to be remunerated for the use of their information in novel ways, an example being micro-deposits of cryptocurrency. In the Kara market, patients whose medical data played a role in training artificial intelligence models get to choose charities to which donations will be made. On Ocean, data is published as a non-fungible token from which tokens for access are created, similar to a cryptocurrency's initial coin offering. A user's crypto wallet then becomes, in effect, a data wallet for a stable commodity currency backed by sets of data [3]. Consequently, Ocean as a coin does not inherit other cryptocurrencies' fiat-like properties.

Discoverability Markets are only attractive to consumers if they can fulfill their data needs, which cannot happen if data sets are difficult to find. Incentives need to exist for sellers to describe their assets appropriately, with the market platform potentially being capable of blending multiple data sets based on metadata [4]. Suppose a customer wants to train their machine learning model on road signs, for instance. In that case, pictures from two separate collections, `EuropeanTrafficImages` and `AmericanTrafficImages`, could automatically be fed into the program, remunerating each seller.

Fairness An exchange should only occur if the seller and sellee concur with the transaction's commodity, policy, and price. A policy may define the data's extent to which it can be used: by whom, for what purposes, and for how long. Trade is fair if the buyer receives what they paid for, the policies are followed, and the seller is remunerated. Since either party involved can withdraw from the sale, fairness leads to both leaving empty-handed in that case [6].

Integrability Data may come from differing sources, contain missing or erroneous information, and be available in a format not directly usable by a consumer. Integrable data has been extracted, transformed, and cleansed. This time-consuming process makes it worth more than raw data. A market should provide incentives for sellers to prepare their assets in this handleable way [4].

Ownership Ownership must be kept track of publicly in a data market to preserve intellectual property rights. Howbeit, traditionally proprietorship of information is in the hands of silos in place of the individuals responsible for generating it. Data exhaust emanating from passive activities such as a purchase on a web store, interactions with smart sensors, and browsing history are monetizable yet do not enjoy the legal protection brought on by copyright laws for active data creation, e.g., writing an email [2]. They are covered by privacy laws instead. Data markets can help establish an economic model ascertaining people control their data through policies.

Provenance Knowing where data came from permits buyers and sellers to audit transactions better. The source may be a factor in determining a data set's quality: inferred data, for instance, is likelier incorrect [2].

Quality Information is prone to change. As it stales over time, its value decreases [2]. Therefore, data correctness is a factor in data markets if the price is automatically discovered.

Security Suppliers of data need to be assured that a leak can not occur. A fair trade in which no other party besides the buyer and seller can see the information (including

2. https://kara.cloud/

doi: 10.2313/NET-2022-11-1_09

intermediaries) is privacy-preserving [6]. However, piracy cannot be prevented if a dishonest buyer receives a full copy of the data. As a result, data escapes need to be prevented by only allowing compute access to sensitive information [3]. Still, the computation cannot occur on a remote machine set up by a cloud operator unless secured computing techniques are employed. Privacy issues and concerns riddle cloud providers. With clients not storing data locally, the attack surface is increased: they may be subject to having their virtual machines cloned or tampered with; audits are burdensome. The cloud provider may subcontract to third parties, making compliance with regulations hard since it is unclear whose responsibility and jurisdiction the data falls under.

Transparency Pricing transparency is necessary for exchanges involving a trade facilitator, i.e., a mediator that may host the marketplace platform. Buyers should be aware of the initial price and the terms of use of the transaction between the seller and the broker [6].

4. Building Blocks

In a secure data market, a seller must convince buyers that they have data the customer needs without revealing it at any point in the trade. Consumers, meanwhile, require assurances that their investment will return the desired results even though they do not know how valuable the data set is in advance [4]. That is fundamentally different from traditional product sales, as the partakers in the exchange are not dealing with a physical good. Instead, a service is provided to a data consumer in which they never get a copy of the information used to produce the final output [5]. Sophisticated technologies come into play to realize this paradigm shift.

4.1. Blockchain

Buying data off silos is problematic since they are in control of an entity that needs to be trusted not to modify it unfairly. A blockchain is a suitable data structure to permanently and irreversibly store the state of a database. Blockchains are immutable, i.e., entries are non-erasable and non-modifiable, meaning that once a transaction is added to the ledger, malicious actors cannot change its contents [7]. Additionally, they are decentrally run and managed, removing the need for an intermediator. As a component of secure markets, blockchains make the entire transaction history traceable, logging accesses and what buyers used it for. Health data usage, for instance, is required to comply with regulations. The chain's transparency lets them track ownership of, e.g., patient records and monitor whether the rules are followed since the ledger's nature ensures entries can only be added but not removed [6]. People that sold Kara their X-rays could see how and where their data is employed in the data economy.

4.2. Smart contracts

Smart contracts are programmable agreements executed on a blockchain-based architecture [8]. In data markets, contracting parties can algorithmically describe the terms of use of private data in the form of a policy [9], such as with whom providers can share banking data. An automatic market maker to establish the price of assets can also be implemented as a smart contract. The contract's code is cryptographically secured and automatically runs once agreed-to conditions are met, e.g., only starting training a supplied machine learning model after the funds have been received. A smart contract can thus act as a trusted trade intermediator in this context [5].

4.3. (Non-fungible) data tokens

Controlling access to data and maintaining intellectual property rights are tasks to be solved in secure data markets such as Kara and Ocean. A naive approach for managing access would be to issue a ticket for users that paid for a service. However, sharing the pass with numerous others, even those who should be barred from possessing one, would be trivial [3]. A mechanism to impose digital scarcity preventing the repeated spending of the same ticket is therefore needed. Blockchain architectures enable this through tokens, whose ownership is tracked and which can either be fungible or not.

A fungible token is identical to others of the same denomination. Holding a data token called `$LabeledTrafficImages`, for example, is a permit that gives the same functionality when using the data services as any other token of that instance. As an analogy, a 1€ coin has equal value to another.

On the other hand, a non-fungible token (NFT) is a digital deed for an asset —– like a collection of labeled traffic images —– that can be stored on the chain, conveying ownership over that property. The non-fungibility comes from the realization that data sets differ, as do physical belongings.

While NFTs could act as data tokens to solve the double-spending issue, the pictures likely interest more than one person. Therefore, the proprietor instead mints an NFT to represent possession of that asset's intellectual property and issues a limited number of licenses (`$LabeledTrafficImages` data tokens) at their discretion to the annotated photos.

Since data tokens can be transferred, a form of identification could additionally be required to redeem the service to combat unrestricted access by people without proper credentials [3].

4.4. Secured data processing

The data market component in which the final output is produced must be secured to prevent intellectual property rights violations and sensitive disclosures of personal digital information. A trusted execution environment offers this degree of protection by computing the result of the buyer's program in a figurative black box [5]. Decrypted data can never be interacted with from the outside by manipulating the data in enclaves, i.e., containers holding the confidential information to be processed and the instructions on how to perform the computation [10]. The application's address space is encrypted in memory [11] and decrypted by the computer's central processing unit.

In some data market architectures, the seller includes the decryption key in the smart contract alongside the use

doi: 10.2313/NET-2022-11-1_09

policy. The key is revealed once the provider's contract verifies that the customer's request satisfies the terms of use [9] [12]. The buyer's smart contract then performs the operations on the raw data inside the trusted execution environment. Alternatively, research and industry standardization efforts are underway [13] for techniques directly performing the computation on encrypted data, namely fully homomorphic encryption.

4.5. Program rewriting and verification

Ensuring policy compliance is an additional challenge requiring a separate module. The motivation of this component is to have a verifier mechanically determine that, given the buyer's program and the data owner's terms of use, the program will not disclose private information once executed. The output is a sound boolean value, meaning that if it certifies the policy is followed, proof of that fact is provided. In case the verifier can not assure that the conditions governing the data's use are met, the module could rewrite the program into one that does so [5].

Say a company wants to price a new product and purchases access to a data set about customers in their target market. They then write a program to query the mean income of students in Bavaria. Intuitively, this is different from asking the average salary of pupils enrolled at the Technical University of Munich with Alice as a first name. While the former question is broad enough to pass the verification step, the latter inquires about an individual and does not bode well with Alice's policy. However, if she is the only person in the data set, both queries are identical.

If removing Alice's record from the data set significantly affects the program's output, the query is said not to be differentially private enough. Differential privacy is a property that algorithms like deep learning models can fulfill that limits how much information concerning their inputs can be revealed [14]. Open-source implementations of differential privacy tools are emerging for general public use, contributing to the adoption of the technique in academia and the industry [15]. To accelerate its prevalence, systems exist that seamlessly integrate with current SQL databases and automatically rewrite queries enforcing differential privacy [16].

5. Architecture

With a secure data market's components and properties now identified, a sample architecture for its realization is provided next. A distinction is made between the stage in which data is added to the market and the one where it is acquired, as they run asynchronously.

1) **Publish step**
a) A seller publishes an encrypted data set to a cloud storage service accessible via a URI that compute jobs may need [9].
b) They provision a smart contract that mints an ERC721[3]-compliant non-fungible token on the Ethereum blockchain pointing to their service [3], claiming themselves as the intellectual property rights holder.
c) The decryption key is secretly kept in the contract with constraints such as the use policy [9].
d) In the desired quantity, the seller's smart contract also mints a pool of ERC20[3] data tokens (licenses) for utilizing the service [3].
2) **Consume step**
a) After identifying relevant data for their purposes in a market's front-end interface, a shopper pays for a data token and writes a smart contract with the code they want to run using the seller's data as input.
b) Once executed, the contract transfers a data token to the seller's crypto wallet as a request to rent access to the data [3].
c) The seller's contract verifies that performing the request will be privacy-preserving (rewriting the request or withdrawing from the sale if necessary) and returns the decryption key for use inside a trusted execution environment [9].
d) The buyer's smart contract runs securely in a trusted execution environment, returning only the computation result.

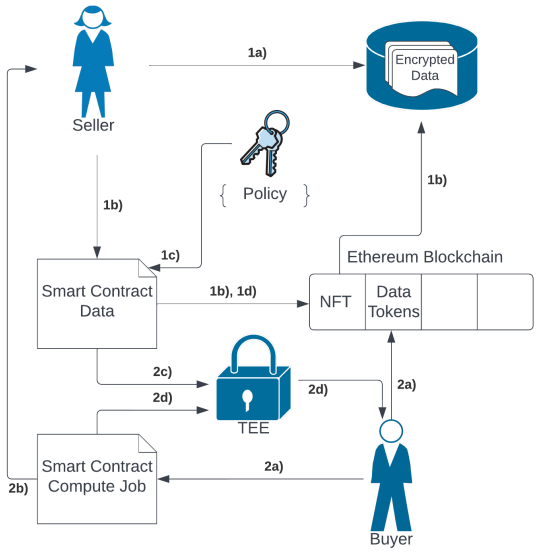


Figure 1: Data sale transaction as outlined in Section 5.

6. Conclusion

In this paper, we established that even though value can be extracted from raw data, little incentive exists for people to curate it in a way that makes it accessible and usable by others. Existing secure data markets, e.g., Kara and Ocean, encourage such behavior while assuring user privacy and control over their information is paramount. Blockchain technologies are fitting in tackling such a task as they enable data trading in a controlled fashion. With smart contracts facilitating data transactions, ensuring adherence to terms of use, and automatic price determination, data assets turn into cryptocurrencies supported by real-world applications.

3. "Ethereum Request for Comment": technical proposal for a standard

doi: 10.2313/NET-2022-11-1_09

References

[1] European Commission, Directorate-General for Communications Networks, Content and Technology, Cattaneo, G., Micheletti, G., Glennon, M., et al., *The European Data Market Monitoring Tool: Key Facts & Figures, First Policy Conclusions, Data Landscape and Quantified Stories: d2.9 Final Study Report*. Publications Office, 2020.

[2] C. Gates and P. Matthews, "Data Is the New Currency," in *Proceedings of the 2014 New Security Paradigms Workshop*, ser. NSPW '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 105–116. [Online]. Available: https://doi.org/10.1145/2683467.2683477

[3] "Tools for the Web3 Data Economy," https://oceanprotocol.com/tech-whitepaper.pdf, Ocean Protocol Foundation with BigchainDB GmbH, Tech. Rep., 2022, last accessed on 2022/05/22.

[4] R. C. Fernandez, P. Subramaniam, and M. J. Franklin, "Data Market Platforms: Trading Data Assets to Solve Data Problems," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 1933–1947, Jul. 2020. [Online]. Available: https://doi.org/10.14778/3407790.3407800

[5] N. Johnson, "Building a Secure Data Market on Blockchain." Burlingame, CA: USENIX Association, Jan. 2019.

[6] P. Banerjee and S. Ruj, "Blockchain Enabled Data Marketplace – Design and Challenges," https://arxiv.org/abs/1811.11462, 2018.

[7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Dec. 2008, last accessed on 2022/06/04. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[8] V. Buterin, "A Next Generation Smart Contract & Decentralized Application Platform," 2015.

[9] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song, "A Demonstration of Sterling: A Privacy-Preserving Data Marketplace," *Proc. VLDB Endow.*, vol. 11, no. 12, p. 2086–2089, Aug. 2018. [Online]. Available: https://doi.org/10.14778/3229863.3236266

[10] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, p. 86, 2016. [Online]. Available: http://eprint.iacr.org/2016/086

[11] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387532

[12] D. Dao, D. Alistarh, C. Musat, and C. Zhang, "DataBright: Towards a Global Exchange for Decentralized Data Ownership and Trusted Computation," 2018. [Online]. Available: https://arxiv.org/abs/1802.04780

[13] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic Encryption Security Standard," Toronto, Canada, Tech. Rep., November 2018.

[14] N. Carlini, C. Liu, U. Erlingsson, J. Kos, and D. Song, "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19. USA: USENIX Association, 2019, p. 267–284.

[15] "The OpenDP White Paper," https://opendp.org/, OpenDP, Tech. Rep., May 2020, last accessed on 2022/06/09.

[16] N. M. Johnson, J. P. Near, J. M. Hellerstein, and D. Song, "Chorus: Differential Privacy via Query Rewriting," *CoRR*, vol. abs/1809.07750, 2018. [Online]. Available: http://arxiv.org/abs/1809.07750

A Case Study of Security Vulnerabilities in Smart Contracts

Marvin James Rautenberg, Filip Rezabek*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: marvin.rautenberg@tum.de, rezabek@net.in.tum.de

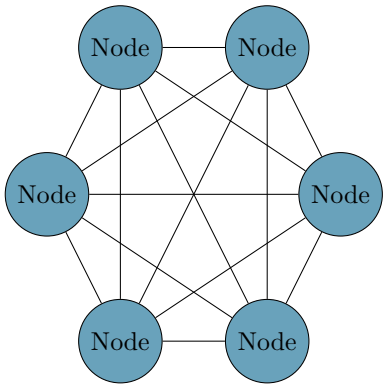
***Abstract*—Ethereum is the first blockchain network that introduced smart contracts which is code that can be executed on a distributed and publicly visible ledger. This makes a trustless and secure system of transaction possible that can not be altered after execution. As a result handling transactions and contracts is significantly improved no matter if the data being processed is tangible or intangible. To ensure this system is appropriate for use in a large scale it is important to analyze the security of it, what possible vulnerabilities the programming language has and how to minimize them which we conclude in a case study that refers to related work and combines all the conclusions. Subsequently we come to the deduction that Turing Completeness is rarely needed in terms of functionality in smart contract programming languages and rather harms the security of it.**

***Index Terms*—ethereum, blockchain, smart contracts, solidity, turing complete**

1. Introduction

Blockchain Technology is steadily growing in popularity and importance posing as one of the most interesting new asset classes on the finance market as even banks now invest into blockchain technology like Bitcoin and Ethereum. Whereas Bitcoin is very limited, Ethereum expanded greatly in the number of use cases it has by introducing the first version of smart contracts that make it possible to run code on a distributed network. This system can eliminate the need for trust in sensitive areas like financial transactions which make transactions much more automated, secure and stable. Having no need for a middle-man to conduct the transaction makes the blockchain a tool to verify and track every transaction that is made on the network.

Even though smart contracts bring multiple advantages compared to traditional contracts, there is the question of how secure this new system is and how it relates to Turing completeness. Additionally the security of the programming language and how to improve the security of the language specifically is important. The paper explains the key concepts needed to understand the analysis in Section 2, then analyze the design of smart contracts and find the connection to security in Section 3. After that we conduct a case study on where the sources of vulnerabilities in the smart contract language Solidity are and how to reduce them in Section 4 and talk about related work in Section 5. Following we come to the conclusion why Turing completeness is rather counterproductive in respect to security in Section 6.



[3]
Figure 1: Blockchain Network

2. Background

2.1. Blockchain

A blockchain is a distributed electronic ledger which records transactions and tracks assets [1]. It is crucial for applications where traditionally you need a trusted middleman to complete sensitive transactions. These transactions can range from a simple currency transaction to law documents and more since a blockchain can track tangible assets like houses or cars, but also intangible assets like intellectual property [1].

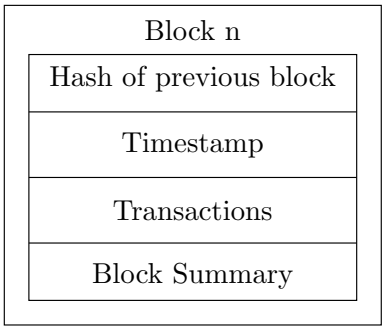
Distributed means it is a decentralized network of nodes like shown in Figure 1. These nodes can be computers running the software of a specific blockchain where every node is connected to each other instead of having a centralized hub of operations like a single server [2].

This means that all the data of the blockchain is publicly visible but the blocks containing the data are not modifiable. Blocks contain the hash of the previous block and multiple transactions in addition to other data like shown in Figure 2.

As this paper will largely focus on Ethereum's implementation of smart contracts we will look further into the attributes of the Ethereum blockchain as other blockchains with the option to create smart contracts are very similar to the Ethereum system.

Transactions on the Ethereum blockchain can only occur between an externally owned address (EOA) and another EOA, between an EOA and a smart contract, or between two smart contracts. An externally owned address usually represents a human made address also called wallet that has a private and public key. A public key is needed to be able to address a specific wallet for a simple

doi: 10.2313/NET-2022-11-1_10



[3]

Figure 2: Block Artchitecture

transaction and the private key is to sign transactions. The private key is used like a PIN on a credit card to confirm a transaction before it is executed. Whereas an EOA stores the balance and nonce which counts the number of all confirmed transactions, a contract-address additionally has code and a storage to track the states it has. This means that a contract address has at least the same amount of actions it can execute as a human created address.

Transactions can not be altered after execution which eliminates the need for trust in a transactions. Usually you have a system in place that needs to be trusted to make sure transactions take place as intended like a bank making sure they deliver your payment and in turn make sure that the seller in a transaction receives his payment from the buyer. Both the seller and the buyer expect the bank to handle everything related to processing the payment and thus have to trust it. The system depends on this trust which is a disadvantage as the trust can be exploited by malicious bankers and the way of making transactions would fall apart if banks became untrustworthy. Smart contracts on the other hand eliminate what would be the bank in a transaction and introduce an electronic contract that is executed on all nodes of the blockchain so everybody can verify that the contract is being executed correctly.

Only one node, the one which completes the contract the fastest actually alters the blockchain by adding the data to the next block of transactions that is chained to the rest of the blocks which in turn permanently alters the whole blockchain for all participants by adding a new block. This process is also called mining. For using the node's computational resources the blockchain rewards the node with the currency the blockchain uses which is Ether in the case of Ethereum. Being able to see all transactions and not being able to change transactions makes the blockchain "highly trustworthy, transparent, and incorruptible" [2]. For all the nodes to agree on certain data values to determine what is a correct output of a transaction there are consensus mechanisms like Proof of Work, Proof of Stake, Proof of Authority and more.

2.2. Smart Contracts

A smart contract is code that is executed in the virtual machine of a blockchain which in our case is the Ethereum Virtual Machine (EVM). The programming language of Ethereum is Solidity which is a Turing Complete programming language. Turing Completeness in the case of Solidity means it can run all programs a Turing Machine can run which mostly differentiates itself from Turing incomplete languages by being able to have complex programs including loops and recursion. The EVM works as a state machine that has an existing state, takes a transaction as an input and combines the two to create a new state. So the blockchain mostly consists of states that are changed over time. A Turing Machine and thus also a turing complete language have the important ability of disregarding the limitations of finite memory which will be an important fact in the analysis in section 3.

3. Smart Contracts Design and Analysis

3.1. Execution of Smart Contracts

You can use an Ethereum Node API to read data from the blockchain. Writing data to the blockchain is much more complicated than reading. You need to send a transaction to the Ethereum Network that specifies which smart contract to alter, which function to execute, any arguments you want to include, and if you are sending any Ether. After signing this transaction it needs a node to accept this transaction which does not have to also execute it. It is possible that the transaction will be forwarded to another node for execution. The transaction will be added to a transaction pool which will be only executed when there are enough transactions to fill a new block. To validate the transaction the transaction is passed as an input to the smart contract which is executed in the EVM. This transaction pool is handled by the miners that all simultaneously execute the transactions in the pool by solving a mathematical problem until one miner finishes all the transactions in the pool and solves the mathematical problem. So only one miner modifies the blockchain and adds a new block and all the other miners are used for verification of the result of the transactions but ultimately discard their calculations. Most of the other blockchains have a similar way of executing smart contracts.

Executing a smart contract is synonymous to buying something from a vending machine [4]. You enter at least the amount of money you need to buy a specific product, you press a button, you get your product and possibly some change back.

Since Solidity, the language Ethereum smart contracts are based on, is Turing complete you can not predict whether a program will finish or not which is why the developers of Ethereum have introduced gas fees to implement some of the benefits of having a Turing incomplete language. Gas fees are payed with every transaction. The person or contract trying to send the transaction has to specify how much they are willing to pay in gas for the transaction to arrive. It is possible to set the limit too low for the transaction to be declined in which case you will still lose the gas fees and the transaction will not be executed. If the gas limit is set appropriately to where the fees do not exceed the limit the transaction will be executed and possible remaining gas that is left over will be reimbursed to the sender of the transaction. Gas fees depend on how busy the Ethereum network is and change over time. Gas fees fix the disadvantage of Turing completeness disregarding the limits of finite memory which could lead to endless looping programs

doi: 10.2313/NET-2022-11-1_10

that never finish as the gas fees will at some point in the execution run out and stop the process. Blocks have an upper gas limit which can not be exceeded by the cumulative sum of the gas fees of the transactions that are in the pool of transactions for the specific block [2]. This ensures that not all transactions will be written into the same block. The nodes of the network can act as miners or EVM depending on the situation [2].

3.2. Other Smart Contract Languages

While Ethereum is the biggest blockchain with the ability to write smart contracts there are several alternatives like Algorand with the programming language TEAL, Cardano which is a blockchain platform using Proof of Stake and EOS.IO which is also a platform built for smart contracts. Algorand tries to solve the problem of scalability, speed of transaction and security that is common among blockchain technology networks [5]. The Ethereum Network can only handle up to 15 Transactions per second [5] whereas Algorand can process up to 1000 transactions per second [6].

The dramatic difference in efficiency is mostly due to Algorand using a different type of consensus mechanism. Reviewing scalablity between different platforms shows a trend of higher transactions per second often being accompanied by weaker security as higher security often implies more resource intensive concepts [5]. Ethereum uses Proof of Work as of the time writing this paper and Algorand uses Proof of Stake which does not require nearly as much resources and scales much better. A switch to Proof of Stake for the Ethereum Network is planned for the future. Algorand's Proof of Stake Mechanism uses the Verifiable Random function to randomly select a Holder of the Algorand currency to validate the next block in the chain instead of using miners like in the Proof of Work approach of Ethereum. A minimum amount of ALGO, the token of the Algorand network, needs to be pledged by a node to be able to validate blocks to ensure that the validator does not act maliciously as it would be unprofitable.

The security advantages TEAL has over Solidity come from it being a Turing incomplete language. Although Solidity is not as limited in the variety of algorithms it can compute like TEAL, being Turing incomplete is the key to reducing the possible attack vectors of the programs that are written with it. [7] concluded that at most 35.36% of smart contracts written in Solidity require Turing completeness to be executable. Therefore the majority does not require it and [7] states that it makes sense to rather use a Turing incomplete language. Although it is also mentioned that a mix of both language types could be the best option to still retain some of the more complex algorithm capabilities of Turing complete languages.

3.3. Security

Since Solidity is the first practical smart contract capable language [8] it does have the benefit of being most popular choice among smart contract developers [5] which makes identifying security vulnerabilities and adopting generalized good practices for coding in Solidity much easier than lesser known languages like TEAL. Major hacks like the DAO hack on the Ethereum network were only possible because of Solidity's Turing completeness, allowing reentrancy attacks causing major financial loss [9]. The EVM was working as intended and the DAO contract itself did not have any flaws but the language itself has flaws due to it being Turing complete which the designers of the language might have overlooked [9].

Most security vulnerabilities are flaws in the coding of the smart contract itself. Some of the most common vulnerabilities are reentrancy attacks and the use of oracle manipulation. Oracles provide data from outside the blockchain. An Oracle could be a sensor on a car tire to monitor the health of the tire to monitor if it is about to break. This kind of information is not on the blockchain as it is real-world data capture by sensors and oracles provide the connection needed to use this external data in smart contracts. Now depending on how sensitive the contract is the choice of oracle can be crucial for the smart contract to work as intended. If an oracle only gets data from one sensor in our example of the car tire, the sensor could be faulty and send wrong data which might trigger a chain of transaction that leads to an emergency call saying a tire broke even though it did not. Using multiple sensors would be needed to make the contract more robust. In other scenarios it is advised to use decentralized points of data sources in an oracle to make sure the data is correct and confirmed by many other sources as this is very important for the contract. Now this also presents an attack vector if you manipulate the oracle you can directly influence the execution of the transactions made by the smart contracts relying on this oracle. This is less a vulnerability of Solidity itself but rather a possible flaw in developing smart contracts made by the developers.

An example for a reentrancy attack could be two people writing each other letters, person A receives a letter from person B, starts writing an answer to the letter from B but does not complete it and starts writing a new letter concerning a different topic and sends it to person B. Now person B answers the letter sent by person A and A finishes writing his first response and sends it to B which would confuse B as it refers to a different conversation that was had before. This type of concept was used in the DAO hack to request Ether multiple times from a smart contract before the contract checked the balance which resulted in the attacker receiving more Ether than intended [10]. Attacks like these can be prevented by better coding practices discussed in section 4.

4. Case Study

4.1. Sources of vulnerabilities

Table 1 shows multiple known vulnerabilities of the Ethereum Network and on what level they appear on according to [11]. We can see that most vulnerabilities can be traced back to Solidity. Considering that the benefits of Turing completeness are not used most of the time in smart contracts made with Solidity it is reasonable to think that the smart contracts would be much more secure if the programming language used was not Turing complete without having to sacrifice too much functionality as the additional functionality of Turing completeness is rarely

TABLE 1: Security Vulnerabilities

	Solidity	EVM	Blockchain
Reentrancy	✓		
Type casts	✓		
Generating Randomness			✓
Gasless send	✓		
Immutable Bugs		✓	
Keeping Screts	✓		
Stack size limit		✓	
Unpredictable state			✓
Call to unknown	✓		

needed. Although it would be much more secure to use a Turing incomplete language we can not neglect the less than 35.36% that [7] concluded to be needing turing completeness. The quantity of the contracts using this complexity does not directly tell the importance of these smart contracts in the network. This 35.36% could be relied on by a lot of other smart contracts that do not need Turing completeness themselves so the influence on the network might be and is probably much higher than the aforementioned 35.36%.

It is important to carefully weigh the benefits of more security versus more functionality and decide which approach is more important in the application of smart contracts to be able to decide if the programming language should be Turing complete. Finding a way to combine both types of languages by having two separate languages that are Turing complete and incomplete to find a middle between the benefits and disadvantages as mentioned in section 3.2 seems to be the best option at the moment.

4.2. Guidelines for writing secure smart contracts

Despite the clear vulnerabilities of a Turing complete language like Solidity, it is possible to minimize the potential security issues in a smart contract by following coding principles. Auditing a smart contract depending on how important it is a good way of reducing the risks of security attacks. We can use static analysis like Slither which is a python program that would directly identify some of the biggest vulnerabilities and warn the developer if his code is prone for issues like Reentrancy. To create safe smart contracts we can not completely rely just on static analysis and need to use manual analysis tools like symbolic execution tools. Echidna is a symbolic execution tool where you can simulate transaction execution without running the code on the public blockchain. This allows us to use Fuzz-Testing to manually assess if functions work as intended. Vulnerabilities like generating randomness where the random values are not as random as the developer wants it to be can just be tested by creating a lot of values with the Fuzz-Testing tool and checking if values are repeated.

5. Related Work

Even though there is literature on similar topics like [12] and [11], they usually only focus on security aspects of smart contracts without connecting the vulnerabilities to the Turing completeness of the language and how this loss or gain in security weighs compared to the functionality. There is literature about the need of Turing completeness in smart contract programming but these mostly are in regard to functionality and do not make a connection to security as well. There is a lot of work regarding blockchain technology, smart contracts in general, how to write smart contracts and most of them refer to Solidity as it is one of the most commonly used languages for smart contracts. There are generally a lot of unscientific guides on how smart contracts work and what practices conclude in a more secure smart contract and what to avoid when programming with Solidity for example. Big attacks on the security of smart contracts are well documented like the DAO hack. This paper rather combines all of these findings and forms a new conclusion.

6. Conclusion

It seems using a Turing complete language has a large negative effect on security as most vulnerabilities can be linked to attributes that only occur in Turing complete languages like a program not terminating by itself. Problems like this can be reduced partly by introducing limitations that are more similar to Turing incomplete languages seen in the introduction of gas fees in the Ethereum network to combat the problem of a program not terminating by itself and thus wasting resources on the network. But it is not always possible to neglect the functionality of Turing complete languages which can provide crucial algorithm support that Turing incomplete languages do not. So either the combination of Turing completeness and incompleteness or using a Turing complete language and using strict security guidelines while creating smart contracts can be viable compromises to ensure security.

Generally the Ethereum Network seems like a very resource intensive network with the use of Proof of Work as the consensus mechanism and the Turing completeness of Solidity which allows for more wasteful and inefficient algorithms compared to a Turing incomplete language like TEAL which also works with a Proof of Stake consensus mechanism that allows the whole Algorand Network to be much more energy efficient than Ethereum. This not only reduces transaction times but also improves scalability and security which seems like an overall improvement.

References

[1] "What is blockchain technology? - ibm blockchain," 2022. [Online]. Available: https://www.ibm.com/topics/what-is-blockchain

[2] R. Modi, *Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and Blockchain*. Packt, 2018.

[3] T. Salman, R. Jain, and L. Gupta, "Probabilistic blockchains: A blockchain paradigm for collaborative decision-making," *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 457–465, 2018.

[4] R. Wilkens and R. Falk, *Smart Contracts: Grundlagen, Anwendungsfelder und rechtliche Aspekte*, ser. essentials. Springer Fachmedien Wiesbaden, 2019. [Online]. Available: https://books.google.de/books?id=k9UyyQEACAAJ

[5] G. A. Tsihrintzis and M. Virvou, "Advances in core computer science-based technologies," Jan 1970. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-41196-1_1

[6] N. Borisov and C. Diaz, *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2021. [Online]. Available: https://books.google.de/books?id=TuVJEAAAQBAJ

[7] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem, "Do smart contract languages need to be turing complete?" *Advances in Intelligent Systems and Computing*, p. 19–26, 2019.

[8] D. Gerard, C. Wagner, K. Boyd, and B. Gutzler, *Attack of the 50 Foot Blockchain: Bitcoin, Blockchain, Ethereum & Smart Contracts*. David Gerard, 2017. [Online]. Available: https://books.google.de/books?id=R7hEDwAAQBAJ

[9] H. HackerNoon, "Should smart contracts be non-turing complete?" Jul 2019. [Online]. Available: https://hackernoon.com/should-smart-contracts-be-non-turing-complete-fe304203a49e

[10] M. Derka, "What is a re-entrancy attack?" Aug 2019. [Online]. Available: https://quantstamp.com/blog/what-is-a-re-entrancy-attack

[11] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," *https://eprint.iacr.org/*, 2016. [Online]. Available: https://eprint.iacr.org/2016/1007.pdf

[12] N. F. Samreen and M. H. Alalfi, "A survey of security vulnerabilities in ethereum smart contracts," *ArXiv*, vol. abs/2105.06974, 2021.

doi: 10.2313/NET-2022-11-1_10

A Brief Overview on HTTP

Justus Wendroth, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge35fup@mytum.de, jaeger@net.in.tum.de

***Abstract*—HTTP has evolved over the last 32 years as new requirements had to be tackled due to the changing use of the web. In this paper, we compare HTTP/1.1, HTTP/2, and HTTP/3 regarding their features and show the limitations they have. Additionally, we compare the versions in terms of latency, packet loss, and usage. We see that higher latency affects HTTP/1.1 worse than HTTP/2. When packet loss increases, the advantages of HTTP/2 decrease and HTTP/1.1 can also be faster. HTTP/3 performs better under high packet loss than HTTP/2. The picture is more mixed for the impacts of latency. HTTP/2 currently appears to be the most widely used version, while the use and adoption of HTTP/3 is increasing.**

***Index Terms*—HTTP/1.1, HTTP/2, HTTP/3, QUIC, latency, packet loss**

1. Introduction

The modern Internet is powered by many technologies: IP addresses for connecting across multiple hops, TCP for reliable data transfer, and TLS for secure data transfer. In this paper, we look at the application protocol HTTP. HTTP is a stateless protocol which was first conceived for sending hypertext (HTML documents). Over the years HTTP evolved and gained new features, including the ability to send data other than HTML. Nowadays, HTTP is not only used for retrieving websites but also to exchange data using REST APIs.

To give a better understanding of HTTP, we briefly look at its history and explore key features introduced with new versions. In Section 2, we also look at the limitations of the three major HTTP versions (HTTP/1.1, HTTP/2, and HTTP/3). For readability, we use h1, h2, and h3 to mean HTTP/1.1, HTTP/2, and HTTP/3 respectively. We then compare the effects of latency and packet loss, and usage for the three major versions in Section 3. In Section 4 we examine related work on comparing HTTP versions. Section 5 concludes this paper.

2. Background

HTTP/0.9, formerly known simply as HTTP, was the first version of the protocol, developed by Tim Berners-Lee at CERN and released in 1990 [1]. HTTP/0.9 only supported GET requests to retrieve resources specified by their path [1]. Due to no HTTP headers being included in this version, only HTML documents could be transmitted [1].

HTTP/1.0 was defined in RFC 1945 in 1996 [2]. This RFC describes the common practices and usages for HTTP at the time as the protocol was extended by different parties and interoperability problems often occurred [1]. One issue with HTTP/1.0 was that a connection could not be reused for multiple requests [1].

HTTP/1.1 (h1) was the first standardized version of HTTP and released in 1997 in RFC 2068 [3]. h1 added new headers, the ability to reuse a connection, pipelining to send multiple request before receiving a response, and other additional functionality.

HTTP/0.9, HTTP/1.0, and HTTP/1.1 could only transfer textual data [2], [3]. HTTP/2 (h2) standardized in 2015 in RFC 7540 is a binary protocol [4]. h2 stems from the Google SPDY[1] project [5]. Additional features include multiplexing of requests (using streams) over a single connection, stream prioritization, and more [4].

HTTP/3 (h3) was standardized in June 2022 [6]. h3 operates on top of QUIC [7], which is a UDP-based transport protocol, instead of TCP. QUIC was first developed at Google [8]. Additional features of h3 and QUIC are the ability for 0-RTT[2] handshakes [7], header compression using QPACK [9], and more.

In the following subsections, we take a closer look at h1 (Section 2.1), h2 (Section 2.2), and h3 (Section 2.3).

2.1. HTTP/1.1

In this section, we look at RFC 2616 from 1999 [10] which was an update of the first h1 version in 1997 [3]. If not otherwise mentioned, we are referring to RFC 2616 [10] in this entire section. One additional feature of HTTP/1.1 is caching, which allows minimizing the number of requests a client has to make to a server as the client can cache previous responses for a certain time. Caching can also be performed by proxy servers, which is another feature of HTTP not discussed further here.

Messages. To transfer data between endpoints (e.g. server and client) HTTP uses HTTP messages. There are two different types of messages. Requests can be used to retrieve data or send data. Responses are the answer to a request. HTTP messages consist of a Request-Line or Status-Line, a list of headers to convey additional information, and the message body with the actual data. Requests have a Request-Line consisting of an HTTP

1. SPDY, QUIC, HPACK, and QPACK are not acronyms but names of projects or standards.
2. RTT = Round-Trip Time

doi: 10.2313/NET-2022-11-1_11

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
Accept: text/html, application/xhtml+xl, application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US, en; q=0.5
Accept-Encoding: gzip, deflate, br
...


200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Keep-Alive: timeout=5, max=1000
Transfer-Encoding: chunked
...

(content)
```

Figure 1: h1 message exchange. Adapted from [1].

method (e.g. GET, POST, PUT, etc.), a Request-URI to indicate the server and resource to be retrieved or sent, and the HTTP version. A Status-Line is composed of the HTTP version, a Status-Code (e.g. 1xx informational, 2xx success, 3xx redirection, 4xx client error, and 5xx server error) to indicate the success of a request, and a Reason-Phrase, which is a textual description of the Status-Code. Status-Lines are used within responses. Fig. 1 shows an example message exchange with h1.

Headers. Headers in HTTP are used to send additional information with a message. There are headers which are required to be sent in requests or responses. Examples of headers are Content-Length to specify the length of the message body, Content-Type to specify the type of resource transported via the message body, Cache-Control to instruct caching mechanisms how to cache resources, and the Cookie headers to add state management to HTTP [11].

Persistent connections. In contrast to HTTP/1.0, h1 allows persistent connections. This means that multiple requests and responses can be sent over a single TCP connection. By not having to open multiple TCP connections for multiple requests overhead is reduced. h1 also introduces pipelining. Pipelining enables sending multiple requests without having to wait for the responses of earlier requests. Responses to pipelined requests must be sent back in the same order in which they were received. Non-idempotent requests (e.g. using the POST method) cannot be pipelined.

Limitations of HTTP/1.1. h1 has several limitations which led to the development of h2 [4]. Pipelining has the problem of head-of-line blocking (HoLB) [12]. HoLB can occur when the first request is for a large file. Sending this large file will cause subsequent responses to be blocked. Pipelining is not widely used by clients and servers as it is difficult to implement [13]. As a solution, multiple TCP connections are opened to be able to send multiple requests at the same time [12]. This causes additional overhead, especially if HTTPS [14] is used. As a consequence, browsers limited the amount of TCP connections per host [12]. A workaround for this is domain sharding, which places resources on different hosts to be able to evade this limitation [12]. Browsers then limited the total number of TCP connections [12]. Other workarounds for h1 include CSS spriting [12].

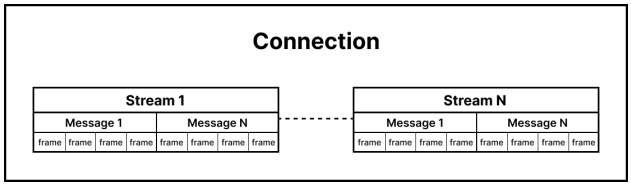


Figure 2: h2 connection. Adapted from [15].

2.2. HTTP/2

h2 builds on top of the core semantics of h1 but introduces multiplexing of requests and responses, which completely changes how data is transferred [4]. In contrast to h1, h2 is a binary protocol, which means that message bodies can be sent in binary format. For most of the new features presented in this section we are referring to RFC 7540 [4]. If this is not the case, we mention this explicitly.

Multiplexing. h2 communication between a client and a server takes place over a single TCP connection (see Fig. 2). A connection is split into multiple streams, where every exchange of request and response is assigned its own stream (see Fig. 2). A request/response exchange fully consumes a stream. Both clients and servers can open new streams. Each stream has an ID, which is assigned by the endpoint initiating the stream and cannot be reused. As can be seen in Fig. 2, Streams are divided into multiple frames. There are different types of frames. For example, the DATA frame carries data, and the HEADER frame is used to open a stream. In every connection there is the stream with ID zero, used for exchanging control messages. In contrast to h1, where most of the information regarding the connection is transferred using headers, h2 conveys a lot of this information using the control stream.

Prioritization. In h2, streams can be dependent on other streams. A dependent always has a lower priority than its parent. When a stream is reprioritized, their dependents move with them. A stream that does not depend on another stream has the stream with ID zero as its parent. If two streams depend on the same stream (can be the zero stream), they can be assigned weights from 1 to 256 and resources should be allocated proportionally to their weight. Assigning priorities is only a suggestion and does not have to be followed by other endpoints.

Server Push. h2 allows a server to push data to a client without a specific request from a client. This can be useful when a client requests a website from the server. Instead of just sending the HTML and waiting for requests of other resources of the website, a server could also push these resources (e.g. CSS and images) directly. Before pushing data to the client, a server needs to send a `PUSH_PROMISE` frame to inform the client about the push.

Flow Control. h2 provides flow control for data frames on the level of individual streams and on the level of the whole connection. Flow control is always specific to connection and cannot be disabled. A receiver can regulate how much data they are able to receive by sending a `WINDOW_UPDATE` frame to the other endpoint.

doi: 10.2313/NET-2022-11-1_11

Header Compression. Header fields (i.e. the name-value pairs in a header) are repetitive and verbose, which causes overhead. Therefore, h2 introduces HPACK [16] for header compression. HPACK uses static and dynamic tables, which are dictionaries that are indexed by indices and contain header fields, to reduce the amount of data to send. The static table is ordered and read-only and provides a list of 61 commonly used headers (e.g. 'accept-encoding: gzip, deflate' has index 16 in the static table) [16, Appendix A]. Dynamic tables are specific to a connection. They are constrained in size and store the header fields encountered during a connection. Additionally, any string can be Huffman encoded using a static Huffman code created for HTTP [16, Appendix B].

Limitations of HTTP/2. h2 has the problem of TCP HoLB, as only one connection is used in h2 [12]. TCP HoLB occurs when one stream has a packet loss, since then all streams have to wait. This is caused by TCP's in-order delivery of packets. h1 does not have this problem, since most of the time multiple TCP connections are used.

Another problem with h2 is the cost of TCP and TLS, since handshakes for TCP and TLS need to be completed to establish a connection [8]. This was also a problem with h1 over TLS, as we have seen previously in Section 2.1. TLS is an optional feature of h2, but most browsers only allow h2 over TLS [12].

2.3. HTTP/3

h3 is the newest version of HTTP [6]. Instead of operating on top of TCP and TLS, h3 operates on top of QUIC [7], which is UDP-based and includes TLS. Using QUIC alleviates the problem of TCP HoLB. h3 includes many of the features introduced with h2. Some of them are implemented directly in h3, while others were moved to QUIC. The semantics of h3 are similar to h1, and h2 [17]. Similar to h2, h3 supports protocol extension. h3 also supports server push [6].

Multiplexing. For this entire subsection, we are referring to [6]. h3 uses QUIC streams for communication, as it has no separate multiplexing mechanism. Similar to h2, a request-response pair consume a stream. To communicate over streams, h3 uses frames similar to h2. There are multiple frame types, including DATA, SETTINGS and HEADERS frames. To exchange control information, h3 uses two separate unidirectional QUIC streams. Two unidirectional streams allow the endpoints to send data as soon as their able to do so after a 0-RTT or 1-RTT connection.

Header compression. In this section, we are citing [9] unless mentioned otherwise. h3 uses QPACK [9] instead of HPACK [16] for header compression. This is because HPACK relies on all frames across all streams being delivered in order. Using HPACK with h3 would therefore cause HoLB, since QUIC does not guarantee in-order delivery across streams but only within a stream. QPACK has similar design goals and concepts as HPACK (e.g. a dynamic table, a static table, and a static Huffman encoding) but uses different mechanisms to achieve this.

QUIC. In this section, we are citing [7] unless mentioned otherwise. To multiplex data QUIC uses streams. Streams are a visible abstraction for application protocols (e.g. h3) operating on to of QUIC. To prioritize streams, QUIC relies on information from the application protocol, as there is no built-in mechanism for exchanging priorities. Streams are split into multiple frames (e.g. STREAM frames to send data or ACK frames to acknowledge packets). For transferring data QUIC uses QUIC packets which consist of multiple frames (from different streams). Packets are sent using UDP datagrams (multiple packets can be in one datagram).

QUIC enables 0-RTT and 1-RTT handshakes by combining the cryptographic (uses TLS 1.3 [18], [19]) and transport handshake. 0-RTT handshakes are possible if there was a prior connection between endpoints. In contrast, h2 over TCP and TLS has a 3-RTT handshake [20].

QUIC connections have ConnectIDs which allow the connection to persist across changes to the underlying IP or port.

Additional features of QUIC include flow control for individual streams and the entire connection, loss detection and recovery mechanisms, and others.

Limitations of HTTP/3. Googles' QUIC implementation consumes twice as much CPU as TCP/TLS [8]. The reasons for the higher CPU usage are the cryptography, the exchange of UDP packets, and maintaining QUIC state. TLS 1.3 enables 0-RTT and 1-RTT cryptographic handshakes [19]. Using TLS 1.3 with h2 reduces the advantages of QUIC over TCP/TLS.

3. Evaluation

In this section, we compare the different HTTP versions regarding their usage and how they react to latency and packet loss.

Effects of latency. Two aspects that have an impact on the latency of the different versions are the handshake they perform (i.e. cryptographic handshake and transport handshake) and their general structure.

h3 performs at most 1-RTT handshakes as the cryptographic and transport handshakes are combined. h1 and h2 over TLS need to perform the TLS handshake and the TCP handshake separately. As mentioned previously, for TLS 1.2 and earlier versions this takes 3-RTT. Therefore, an increase in latency should affect the h3 handshake less than h1 and h2. Latency should affect h2 less than h1 because when h1 is used, often multiple connections are opened and for each connection the handshake has to be repeated. Langley et al. [8] show that the handshake latency increases linearly for h2 over TCP/TLS and remains almost constant for Google QUIC.

Since h2 and h3 support multiplexing of requests and responses, they should handle latency better than h1. References [12], [21] show that h2 reacts better to latency than h1. Trevisan et al. [22] show the order h3, h2, and h1 from best to worst. In contrast, Saif et al. [23] show h3 performing worse than h2 regarding QoE (Quality of Experience) measurements using Lighthouse. This is because in their tests LCP (i.e. Largest Contentful Paint: time for the largest payload to be rendered completely)

performed consistently much worse for h3 than h2. In addition, no 0-RTT handshakes were used for h3.

Effects of packet loss. h2 has the problem of TCP head of line blocking, where a packet loss affects all streams which are multiplexed over a single TCP connection. Therefore, packet loss might affect h2 worse than h1. h3 solves the TCP HoLB by using QUIC over UDP instead of TCP. It should therefore react better to packet loss than h2. De Saxcé et al. [12] show decreasing performance improvements for h2 compared to h1 with higher rates of packet loss. h1 can also perform better for higher packet loss. In contrast to this, Corbel et al. [21] show that h2 performs better than h1 under high packet loss rates. This might be because they used h1 pipelining over a single TCP connection which would also be affected by TCP HoLB. h3 shows better performance than h2 at higher packet loss rates [22], [23].

Usage. In this section, we mostly focus on data provided by Cloudflare, w3tech and the HTTP archive, as data mentioned in papers quickly becomes outdated. For example, Trevisan et al. [22] measured the adoption of h3 at 4.8% in October 2020, but as we show in the following subsection, the adoption of h3 has significantly increased over the last 1.5 years.

For the 30 days prior to June 11, 2022, Cloudflare radar showed that the traffic passing through their infrastructure was 8% h1, 68% h2, and 24% h3 [24]. Additionally, over the 12 months prior to June 2022, h2 made up the majority of requests for Cloudflare customer content and the number was increasing [25]. h1 requests were stable, while h3 requests increased and surpassed h1. W3tech measures the adoption of h3 and h2 by websites by examining the top 10 million websites from Alexa[3]and 1 million from Tranco[3] [26]. When a technology is found on a website, that website adopts the technology. h2 is adopted by 45.8% of websites, which has stayed relatively stable over the 12 months prior to June 2022 [27]. h3 is adopted by 25.2% of websites, which has increased over the 12 months prior to June 2022 [28]. The HTTP archive crawls millions of URLs from the Chrome User Experience Report on a monthly basis [29]. For June 1, 2022, h3 support was at 15% for desktop and 15.3% for mobile and has been steadily increasing since May/June 2020. The number of h2 requests on June 1, 2022, was at 67.7% for desktop and 67.8% for mobile and also steadily increasing.

It is difficult to compare data from different sources. However, a general trend in increasing usage of h2 and increasing adoption and usage of h3 can be seen. h1 is also still widely used. For example, the Facebook bot, Google bot, and LinkedIn bot, which are crawlers for the respective social media sites and search engines, perform a lot of their requests using h1 [25].

4. Related work

De Saxcé et al. compare h1 and h2 using page load time as the performance indicator [12]. They clone 20 popular websites to a server and use a LAN connection to that server to test the impact of latency and packet loss on h1 and h2. They show that an increase in latency impacts page load times for h1 more than h2. In contrast, h2 shows less performance benefits the higher the packet loss is. h2 can also take longer than h1 for higher packet loss. Some limitations include not using TLS and no domain sharding, as all data is on a single server.

Saif et al. compare h2 (over TCP and TLS 1.3) and h3 using the Lighthouse open-source tool, which measures performance and QoE (Quality of Experience) aggregating different metrics to give a performance score [23]. Their test setup is a server in a virtual machine and Google Chrome as the client on the same machine. They use web content to resemble a realistic web page. To compare effects of latency and packet loss, they increase these parameters individually (latency to 1000ms and even 2000ms; packet loss to 1.4%). The baseline score without any adjustments to latency or packet loss was 65 for h3 and 87 for h2. h3 showed consistently worse performance for increases in latency. For packet loss, h3 performed better than h2 for a packet loss starting at 1% and the score stayed relatively flat in contrast to h2. h3 had consistently worse scores for LCP (i.e. Largest Contentful Paint: time for the largest payload to be rendered completely) which makes up a large percentage of the Lighthouse score. One limitation is that no 0-RTT connection establishments were used for h3.

Trevisan et al. look at the adoption and performance of h3 [22]. Using a data set of 5 million URLs, they found that 4.8% of these websites supported h3 in October 2020. Of the websites that support h3, 51% still retrieve one or more objects using h1. For testing performance they make requests to a subset of websites supporting h3. They increase latency to 200ms, increase packet loss to 5% and decrease bandwidth to 1 Mbit/s separately. To compare the effects of varying these parameters, they used onLoad (when all elements of a webpage have been loaded and parsed) and speedIndex (when visible portions of the page are displayed). They compared h1, h2, and h3 regarding latency and found that the mean onLoad and speedIndex times are best for h3, second for h2, and worst for h1. The improvements increase with higher latencies. They also compare h2 and h3 regarding bandwidth and packet loss and found that h3 had better performance at low bandwidth, but performance was similar for high packet loss. They always use a fresh browser profile with empty cache and no pre-existing connections. This means no 0-RTT for h3.

A limitation of all papers presented here is that they only compare two different HTTP versions (h1 and h2 or h2 and h3). Trevisan et al. have measurements across all versions, but they mostly focus on comparing h2 and h3 [22].

5. Conclusion

This paper gave an overview of the different HTTP versions HTTP/1.1 (h1), HTTP/2 (h2), and HTTP/3 (h3) and their most important features. In h1, messages are exchanged via requests and responses and headers can be used to convey additional information. h2 builds on these features from h1 by including mechanisms to multiplex

3. Alexa [30] and Tranco [31] provide top sites rankings

doi: 10.2313/NET-2022-11-1_11

requests and compress headers. h3 builds on the multiplexing from h2 by using QUIC and UDP instead of TCP and therefore alleviating the TCP HoLB problem of h2.

We then compared h1, h2, and h3 in terms of how they perform under different amounts of latency and packet loss. We also compared the usage of the different versions. h2 performs better than h1 at high latency. For higher packet loss, the benefits of h2 are reduced and h1 can also perform better. By comparing h3 and h2, we see that h3 performs better under higher packet loss. For higher latency, h2 performs better for some tests and h3 for others. h2 currently seems to be the most used version (June 2022). The use and adoption for h3 is increasing. h1 usage is relatively stable.

In the future, there is room for a comparison of all three HTTP versions regarding latency, packet loss, bandwidth, and maybe other parameters, since most papers only compare two different HTTP versions (h1 and h2 or h2 and h3).

References

[1] "Evolution of HTTP - HTTP | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[2] H. Nielsen, R. T. Fielding, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.0," Internet Engineering Task Force, Request for Comments RFC 1945, May 1996, num Pages: 60. [Online]. Available: https://datatracker.ietf.org/doc/rfc1945

[3] R. T. Fielding, H. Nielsen, J. Mogul, J. Gettys, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Engineering Task Force, Request for Comments RFC 2068, Jan. 1997, num Pages: 162. [Online]. Available: https://datatracker.ietf.org/doc/rfc2068

[4] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Engineering Task Force, Request for Comments RFC 7540, May 2015, num Pages: 96. [Online]. Available: https://datatracker.ietf.org/doc/rfc7540

[5] "SPDY." [Online]. Available: https://www.chromium.org/spdy/

[6] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet Draft draft-ietf-quic-http-34, Feb. 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-http

[7] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Request for Comments RFC 9000, May 2021, num Pages: 151. [Online]. Available: https://datatracker.ietf.org/doc/rfc9000

[8] 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, Aug. 2017, pp. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[9] C. B. Krasic, M. Bishop, and A. Frindell, "QPACK: Header Compression for HTTP/3," Internet Engineering Task Force, Internet Draft draft-ietf-quic-qpack-21, Feb. 2021, num Pages: 51. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-qpack

[10] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Engineering Task Force, Request for Comments RFC 2616, Jun. 1999, num Pages: 176. [Online]. Available: https://datatracker.ietf.org/doc/rfc2616

[11] A. Barth, "HTTP State Management Mechanism," Internet Engineering Task Force, Request for Comments RFC 6265, Apr. 2011, num Pages: 37. [Online]. Available: https://datatracker.ietf.org/doc/rfc6265

[12] H. de Saxcé, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?" in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2015, pp. 293–299.

[13] M. Nottingham, "Making HTTP Pipelining Usable on the Open Web," Internet Engineering Task Force, Internet Draft draft-nottingham-http-pipeline-01, Mar. 2011, num Pages: 10. [Online]. Available: https://datatracker.ietf.org/doc/draft-nottingham-http-pipeline-01

[14] E. Rescorla, "HTTP Over TLS," Internet Engineering Task Force, Request for Comments RFC 2818, May 2000, num Pages: 7. [Online]. Available: https://datatracker.ietf.org/doc/rfc2818

[15] "HTTP/1.1 vs HTTP/2: What's the Difference? | DigitalOcean." [Online]. Available: https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

[16] R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," Internet Engineering Task Force, Request for Comments RFC 7541, May 2015. [Online]. Available: https://datatracker.ietf.org/doc/rfc7541

[17] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," Internet Engineering Task Force, Internet Draft draft-ietf-httpbis-semantics-19, Sep. 2021, num Pages: 252. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-semantics

[18] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, Request for Comments RFC 9001, May 2021, num Pages: 52. [Online]. Available: https://datatracker.ietf.org/doc/rfc9001

[19] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018, num Pages: 160. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446

[20] "Introducing Zero Round Trip Time Resumption (0-RTT)," Mar. 2017. [Online]. Available: http://blog.cloudflare.com/introducing-0-rtt/

[21] R. Corbel, E. Stephan, and N. Omnes, "HTTP/1.1 pipelining vs HTTP2 in-the-clear: Performance comparison," in *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)*, Jul. 2016, pp. 1–6, iSSN: 2162-190X.

[22] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, "Measuring HTTP/3: Adoption and Performance," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, Jun. 2021, pp. 1–8.

[23] D. Saif, C.-H. Lung, and A. Matrawy, "An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021, pp. 1–6, iSSN: 1938-1883.

[24] "Cloudflare Radar." [Online]. Available: https://radar.cloudflare.com

[25] "HTTP RFCs have evolved: A Cloudflare view of HTTP usage trends," Jun. 2022. [Online]. Available: http://blog.cloudflare.com/cloudflare-view-http3-usage/

[26] "Web Technologies Statistics and Trends." [Online]. Available: https://w3techs.com/technologies

[27] "Usage Statistics of HTTP/2 for Websites, June 2022." [Online]. Available: https://w3techs.com/technologies/details/ce-http2

[28] "Usage Statistics of HTTP/3 for Websites, June 2022." [Online]. Available: https://w3techs.com/technologies/details/ce-http3

[29] "HTTP Archive: State of the Web." [Online]. Available: https://httparchive.org/reports/state-of-the-web

[30] "End of Service Notice." [Online]. Available: https://www.alexa.com/

[31] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, Feb. 2019.

Deterministic Networking - DetNet

Berdiguly Yaylymov, Filip Rezabek*, Kilian Holzinger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: berdiguly.yaylymov@tum.de, rezabek@net.in.tum.de, holzinger@net.in.tum.de

***Abstract*—Deterministic networking is rapidly gaining importance, as several network applications and industries demand deterministic network services. Several of those network applications demand low end-to-end latencies and low packet loss. The IETF Deterministic Networking architecture provides network-layer services and IEEE 802.1 Time-Sensitive Networking provides data link-layer services. Both DetNet and TSN play a critical role in providing real-time low latency and deterministic services for the next-generation networks. In such context, this paper presents a broad overview of DetNet and summarizes its key features.**

***Index Terms*—Deterministic Networking (DetNet), Time Sensitive Networking (TSN), Ultra-Low Latency (ULL)**

1. Introduction

Traditional Ethernet does not fulfill the communication requirements of critical real-time systems such as aerospace, industrial automation, automobiles, etc., which require high-bandwidth and low delay for communication networks with the increasing communication data traffic. Therefore, industries and customers turn their attention to Quality of Service (QoS) metrics and Ultra-Low Latency (ULL) paradigms, which provide end-to-end latencies in a matter of a few microseconds and milliseconds, depending on the applied applications [1] [2].

The IEEE 802.1 Time-Sensitive Networking Task Group (TG) and the IETF Deterministic Networking Working Group (WG) are collaborating [2] to establish a common architecture for Layers 2 and 3 of the OSI[1] model. DetNet focuses on Layer 3 routed segments, whereas TSN focuses on Layer 2 bridged networks [2]. Their main goal is to provide support for high reliability in packet delivery, deterministic worst-case bounds on latency [2] and better worst-case QoS metrics for best-effort flows [3]. This survey is intended to provide a comprehensive overview of the IETF DetNet and its features and discuss its current state and future.

The rest of this paper is organized as follows. Section 2 deals with some background studies on Deterministic Networking and how it emerged and gained interest. Section 3 provides an overview of IEEE TSN and TSN-related studies and focuses on the data link layer. Section 4 covers an overview of IETF DetNet, its architecture, flow types and practical use cases and focuses primarily on the network layer. Section 5 illustrates the existing DetNet and TSN standards and studies and provides some similarities and differences between both paradigms, and tries to summarize their key features. Previously published related work and research studies on this article are provided in Section 6. Finally, in Section 7 we discuss and review practical problems in DetNet and conclude this proceeding with future research directions.

1. OSI - Open Systems Interconnection

2. Background

The increasing demand for ultra-low latency networking standards led to the development of a unified data link-layer protocol by the IEEE 802.1 WG called Audio Video Bridging (AVB) in 2005 [4]. AVB ensured real-time requirements such as the transmission of audio and video streams but lacked fault-tolerance to enhance its reliability [5] and was subject to some system failures and malicious cyber attacks [4].

Consequently, in 2012, the IEEE 802.1 WG expanded the current AVB and renamed it to **Time-Sensitive Networking (TSN)**. The TSN enhances time synchronization, supports the scheduling of real-time time-sensitive data streams, and improves the streams' reservation ability [4]. Together with this expansion, the networks were getting larger, requiring deterministic networks. For example, public infrastructures such as electricity automation require deterministic paradigms over a wide area, whereas the TSN provides support for Layer 2 control systems and cannot support structures beyond LAN boundaries. Therefore, Layer 3 networks were required without losing Layer 2 capabilities [6].

Motivated by these shortcomings, the IETF, in cooperation with Standards Development Organizations (SDOs) and IEEE 802 developed the **Deterministic Networking (DetNet) WG** in 2015 [7]. A key feature of DetNet is the ability to establish a multi-hop path over the IP network with a particular flow and ultra-low jitter and low latency [6].

In general, latency refers to a time delay in an end-to-end packet delivery between a sender and a receiver. Thus, the term ultra-low latency usually refers to latencies that have speeds under 1 millisecond. The term bounded latency is often used in ULL systems and describes time delay that must not exceed some predetermined value, e.g., to ensure the appropriate functionality in automation systems.

Jitter refers to variations of packet latencies, which are often caused by congestion. Therefore, two key QoS metrics of ULL networking are jitter and latency.

doi: 10.2313/NET-2022-11-1_12

3. Overview of Time-Sensitive Networking

The IEEE 802.1 TSN TG standards and services extend the Ethernet data-link layer and guarantee data transmission with ultra-low latency and jitter [2]. The TSN is based on a best-effort packet network consisting of bridges and network appliances.

3.1. TSN Features

Time synchronization: Time synchronization is accomplished using IEEE 1588 Precision Time Protocol (PTP) configuration, which is, e.g., a stand-alone standard IEEE 802.1AS [8]. All of the devices in a network can synchronize their internal clocks with an accuracy of up to 10ns.

Contracts between transmitters: Each TSN flow functions with a contract between the transmitter of the flow and the network. Therefore, such features are provided:

- **Zero congestion loss and bounded latency:** Congestion and packet loss are caused by the overflowing streams in the network node. These shortcomings are eliminated thanks to buffer allocation and queuing algorithms [2]. Buffer allocation is accomplished through computing worst-case buffer requirements. There are a couple of queuing algorithms defined in IEEE Std 802.1Q, which are: Credit Based Shaper (CBS), Time-Scheduled Queues, Transmission Preemption and Asynchronous Traffic Shaping (ATS).
- **Ultra reliability:** Equipment failure is also one of the main reasons for packet loss. The main method of improving the reliability of TSN networks is Frame Replication and Elimination for Reliability (FRER) [9].

A sample FRER is illustrated in Figure 1. Packets can be both replicated and eliminated at each node of the TSN Timing Model [7].

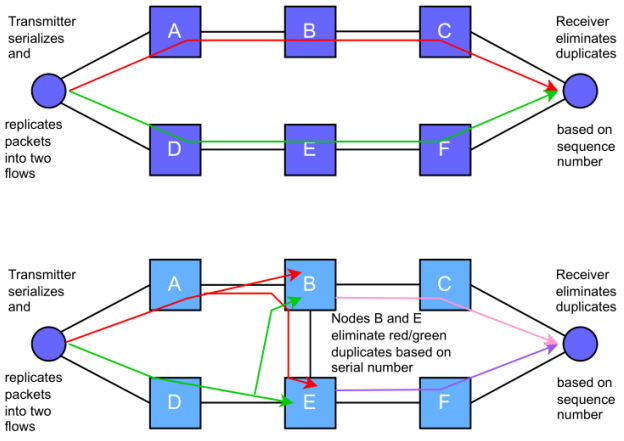


Figure 1: Packet replication and elimination [7]

3.2. TSN Use Cases

TSN use cases are similar to DetNet use cases and are thoroughly explained in [7]. Some of them are:

- Professional audio and video studios.
- Electrical power generation and distribution.
- Cellular radio.
- Automotive and other vehicle applications.

4. Overview of Deterministic Networking

In this section, a detailed overview of the IETF Deterministic Networking (DetNet) WG will be described. According to [2], the IETF DetNet WG collaborates with IEEE 802.1 TSN TG to define a common architecture for Layers 2 and 3. DetNet is considered to be a representative wide-area networking technology. To overcome the limitations of the LAN-based narrow-area networking technologies, such as TSN, IP/MPLS[2]-based wide-area networking technology is being standardized [10]. Like TSN, DetNet's main goal is to support deterministic worst-case bounds on latency, jitter, zero/low packet loss and reliability.

4.1. DetNet Architecture

DetNet data plane and functionality are composed of two sub-layers: DetNet service sub-layer and DetNet forwarding sub-layer. Each one of these layers is classified according to the DetNet flow. The service sub-layer provides service protection functions and classifies time-determined flows. The service protection function duplicates and delivers packets through several packets, and deletes the duplicated packets according to their sequence number [10]. The forwarding sub-layer provides explicit routes and resource reservations for time-determined flows, which are the basis of wide-area networks. Note that these sub-layers are helpful, but not mandatory to implement and should not be considered a formal requirement. Some technologies are still capable of providing DetNet services, even if they do not adhere to this strict sub-layer division. The illustration of the DetNet data-plane layering model is presented in Table 1. Not all sub-layers are required for a particular network or a particular application.

TABLE 1: DetNet Data-Plane Protocol Stack [11]

Sub-layers	Source	Destination
Service sub-layer	Packet sequencing Flow replication Packet encoding	Duplicate elimination Flow merging Packet decoding
Forwarding sub-layer	Resource allocation Explicit routes	Resource allocation Explicit routes
	Lower layers	Lower layers

Application: Source and Destination are considered to be any application that is going through the stack.

Packet sequencing: This sub-layer supplies the sequence number for packet elimination and replication. It is not needed if a higher-layer protocol performs the sequencing.

Duplicate elimination: This sub-layer discards all duplicate packets generated by DetNet flow replication based on the specified sequence number. It can also resequence the packets to restore the order of the packets, which may be disrupted by the loss of packets on multiple paths [11].

2. IP/MPLS - Internet Protocol/Multiprotocol Label Switching: routing system that enables fast data switching

doi: 10.2313/NET-2022-11-1_12

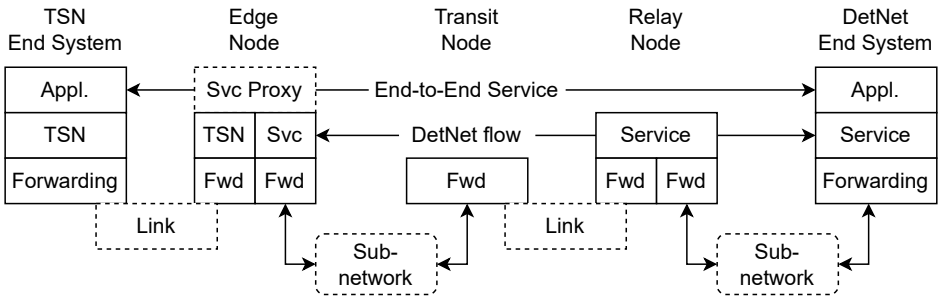


Figure 2: A simple DetNet-enabled network [11]

Flow replication: This sub-layer is part of DetNet service protection. Packets belonging to DetNet compound flow are replicated, apart from packet sequencing, into several DetNet member flows. This replication may also be performed using techniques such as multicast replication but with resource allocation implications [11].

Flow merging: The functions for flow merging combine DetNet member flows together for packets coming up the stack. This sub-layer performs packet replication and elimination, together with packet sequencing, duplicate elimination and flow replication [11].

Packet encoding and decoding: These sub-layers take packets from different DetNet member flows. Packet encoding combines the information and transmits them to different DetNet member flows. Packet decoding computes the original packets and transmits them to different DetNet member flows.

Resource allocation: Providing paths for DetNet flows, queuing, and shaping mechanisms are usually provided by this sub-layer.

Explicit routes: These are arrangements of fixed paths, based on the DetNet forwarding sub-layer that is specified in advance to avoid the effects of network convergence on DetNet flows [11].

A simple concept of a DetNet network is illustrated in Figure 2. In this figure, "Fwd" and "Forwarding" refer to the forwarding sub-layer, "Svc" and "Service" refer to the service sub-layer [11].

4.2. DetNet Flow Types

Depending on the type of end systems, DetNet flow may have different formats. According to the end system types, the following four types of a DetNet flow are distinguished:

App-flow: The native data (payload) flows between the DetNet source and destination end systems.

DetNet-s-flow: This flow contains the DetNet-related specific attributes that provide services for elimination and replication functions. This flow is a specific data flow format that requires the service protection feature and is bound to the service sub-layer.

DetNet-f-flow: This is also a specific format of a DetNet flow. This flow is bound to the forwarding sub-layer and contains specific attributes that provide services for congestion protection.

DetNet-sf-flow: This is a specific data flow format, which signals the forwarding function during forwarding. This flow is bound to the both service and forwarding sub-layers of the DetNet stack model.

4.3. DetNet Use cases

DetNet is not considered to be "a new kind of network", but is supported by Ethernet extensions, including elements of TSN and related standards. There are several use cases in [12], which explains the type of these use cases, their future, and what IETF should deliver to enable them. DetNet shares same use cases [12] as TSN, including some additional cases:

- Building automation systems (BASs)
- Industrial machine to machine (M2M)
- Private blockchain
- Mining industry
- Network slicing

However, there are various use cases, which were considered by DetNet WG and the Design Team to be out of the scope of DetNet. The scope of DetNet networks is limited to services that can be centrally controlled, e.g., corporate networks. From this point of view, "the open Internet" is excluded from DetNet networks. Maintaining a high-quality user experience and low latency is critical for the use cases listed below. Due to jitter and time delay, these use cases, when run over the open Internet, are considered to be outside the scope of DetNet [12]. These use cases are:

- Media content delivery
- Online Gaming
- Virtual Reality

Nevertheless, we provide a detailed overview of the two applications that fall under the scope of DetNet from the use cases listed above.

Building Automation Systems (BASs): Building Automation Systems manage devices and sensors in a facility to improve the comfort of occupants, reduce energy consumption and respond to emergencies and failures [12]. For instance, the BAS controls the heating, ventilation, and air conditioning to maintain and reduce energy consumption. The basic architecture of BAS is shown in Figure 3. The BAS network has two layers: the upper layer – management network and the lower layer – field network. IP-based communication protocols are used in the upper layer, while in the lower layer, non-IP-based

doi: 10.2313/NET-2022-11-1_12

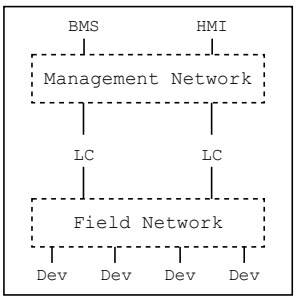


Figure 3: BAS Architecture [12]

communication protocols are used. Management networks can be best effort, whereas field networks have particular timing requirements.

Private Blockchain: Blockchain has spread far beyond its original host into several other industries. These industries are logistics, security, smart manufacturing, legal rights, and others. Designated and carefully managed networks of these industries, in which blockchain runs, may require deterministic networking. These kinds of implementations are called "private blockchains". Blockchain operation could be much more efficient if DetNet services were available to reduce packet loss and latency [12].

Currently, private blockchain runs in Layer 2 or Layer 3 VPNs without guaranteed determinism. Industries have realized that implementing and improving determinism in their blockchain networks would also improve the performance of their service, because low latency would speed up the consensus process. Some of the private blockchain requests [12] to the IETF are listed below:

- Layer 2 and 3 multicasts of blockchain traffic
- Item and block delivery with low latency and low packet loss
- Coexistence of IT traffic and blockchain in a single network

5. Comparison of DetNet and TSN

There are several differences and similarities between these two standards. The main difference between DetNet and TSN is the layering in the OSI model. DetNet operates on the Layer 3 protocols whereas TSN is confined to Layer 2.

The data plane of these standards is also different. DetNet nodes can connect to other subnetworks, such as Optical Transport Network (OTN) and MPLS Traffic Engineering. TSN cannot achieve multi-layer systems, while DetNet can. However, TSN and DetNet share the same features such as time synchronization, frame replication and elimination.

DetNet has to deal with more security challenges than TSN because it operates on Layer 3 networks and in open environments, which results in more security threats. As a result, DetNet focuses on and offers more security solutions than TSN. An example of such an attack is a man-in-the-middle attack, which can impose and adjust delays in a node and undermine a real-time application [13].

6. Related Work

There are a couple of surveys on Deterministic Networking in [6] [12] [11]. An IEFT draft of the problem statement on deterministic networking has been presented in [6]. A broad survey about the use cases of deterministic networking and its overall architecture has been provided in [12] and [11] respectively. A sample *DetNet Simulator* based on OMNET++ and NeSTiNG, which overcomes some limitations such as allowing simulations of the full DetNet/TSN protocol stack, has been presented in [3]. A broad survey of the Audio and Video Bridging (AVB) standard, the predecessor of TSN, has been introduced in [4] and in [5]. An introduction to Time-Sensitive Networking and its essential features has been provided in [7]. Furthermore, an up-to-date comprehensive survey of the studies that specifically target the support of ULL in 5G networks, DetNet and TSN has been presented in [2] and in [10].

Many applications are likely to use techniques to increase the probability that a particular packet will be delivered. And when topology-fixed paths are used, which are protected against congestion loss a Frame Replication and Elimination for Reliability standard can guarantee a substantial reduction in the probability of packet loss than any other standards. Therefore, a survey on Frame Replication and Elimination for Reliability has been provided in [9].

7. Discussion and Conclusion

One of the future challenges of Deterministic Networking is packet replication and elimination (PRE). DetNet can ensure reliability, the increase in the probability of packets reaching their destination and the overall reduction of end-to-end latency [14] through packet replication and elimination. However, the increase in effective bandwidth required for a DetNet flow is a major disadvantage of PRE. This can be overcome by reducing the replication level, but it can affect the reliability and thus impact the balance between packet replication and bandwidth [2]. Nevertheless, the balance must be ensured for the operation of the DetNet.

This paper provided a broad overview of deterministic network management and control systems, which can operate on DetNet components and provide ULL services and features such as low jitter, low congestion loss, reliability and time synchronization. This survey also discusses the differences and similarities between DetNet and TSN standards and provides their features and use cases. Despite their shortcomings and numerous limitations, there is a need for an extensive evaluation of both DetNet and TSN standards. A combination of these standards has a high chance of impacting the traditional Ethernet networks and providing effective low latency services to users.

References

[1] J. Prados-Garzon, T. Taleb, and M. Bagaa, “LEARNET: Reinforcement Learning Based Flow Scheduling for Asynchronous Deterministic Networks,” pp. 1–6, 2020.

doi: 10.2313/NET-2022-11-1_12

[2] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, Firstquarter 2019.

[3] V. Addanki and L. Iannone, "Moving a step forward in the quest for Deterministic Networks (DetNet)," pp. 1–9, 2020.

[4] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A Survey of Real-Time Ethernet Modeling and Design Methodologies: From AVB to TSN," *ACM Computing Surveys*, vol. 55, no. 2, pp. 1–36, 2022.

[5] O. Kleineberg, P. Fröhlich, and D. Heffernan, "Fault-tolerant Audio and Video Bridging (AVB) Ethernet: A novel method for redundant stream registration configuration," *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2012.

[6] N. Finn and P. Thubert, "Deterministic Networking Problem Statement," pp. 1–11, 2019.

[7] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018.

[8] K. B. Stanton, "Distributing deterministic, accurate time for tightly coordinated network and software applications: IEEE 802.1AS, the TSN profile of PTP," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 34–40, 2018.

[9] I. L. S. Committee, "IEEE Standard for Local and metropolitan area networks-Frame Replication and Elimination for Reliability," *IEEE Communications Standards Magazine*, pp. 1–102, 2017.

[10] E. Kim, Y. Ryoo, B. Yoon, and T. Cheung, "Active control and management system for providing the ultra-low latency service on deterministic networks," *2021 Twelfth International Conference on Ubiquitous and Future Networks*, pp. 70–74, 2021.

[11] N. Finn, P. Thubert, B. Varga, and J. Farkas, "Deterministic Networking Architecture," pp. 1–38, 2019.

[12] E. Grossman, "Deterministic Networking Use Cases," pp. 1–97, 2019.

[13] T. Mizrahi, "Security Requirements of Time Protocols in Packet Switched Networks," pp. 1–36, 2014.

[14] J. De Armas, P. Tuset, T. Chang, F. Adelantado, T. Watteyne, and X. Vilajosana, "Determinism through path delivery: Why packet replication makes sense," *2016 International Conference on Intelligent Networking and Collaborative Systems (InCoS)*, pp. 150–154, 2016.

doi: 10.2313/NET-2022-11-1_12

ISBN 978-3-937201-76-4
DOI 10.2313/NET-2022-11-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)