Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

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

Winter Semester 2021/2022

Munich, Germany

Editors

Georg Carle, Stephan Günther, Benedikt Jaeger

Publisher

Chair of Network Architectures and Services

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

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

Winter Semester 2021/2022

Munich, July 30, 2021 – February 27, 2022

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET NET-2022-07-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Winter Semester 2021/2022

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 WS 21/22
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, July 30, 2021 – February 27, 2022
ISBN: 978-3-937201-75-7

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-NET-2022-07-1
Innovative Internet Technologies and Mobile Communications (IITM) NET NET-2022-07-1
Series Editor: Georg Carle, Technical University of Munich, Germany

Preface

We are pleased to present to you the proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM) during the Winter Semester 2021/2022. 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 Jakob Weigand with the paper *Position-based Routing in Flying Ad Hoc Networks* and Oliver Lemke with the paper *Survey on Machine Learning-based Autoscaling in Cloud Computing Environments* .

Some of the talks were recorded and published on our media portal `https://media.net.in.tum.de`.

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, May 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

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

Sayantini Majumdar (sayantini.majumdar@tum.de)
Technical University of Munich

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

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

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

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

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

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

Seminar Homepage

`https://net.in.tum.de/teaching/ws2122/seminars/`

Contents

Block Seminar

Seminar

State of the Art of DDoS Mitigation Techniques

Franz Josef Ennemoser, Patrick Sattler* and Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: franzjosef.ennemoser@tum.de, sattler@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—Distributed Denial of Service (DDoS) attacks continue to be one of the biggest threats for online services. This has created a large demand for DDoS Protection Services (DPS) in the last decade, who use their clouds to defend customers from larger attacks every year. Since these types of attacks are launched from many different sources, preventing or mitigating DDoS attacks requires sophisticated defence mechanisms. The paper shows the three basic components of a potent defence against DDoS attacks which are attack detection, traffic classification and attack response. While the defence mechanisms of DPS providers are proprietary, we showcase some mechanisms that demonstrate how DPS systems can be comprised in practice. Furthermore, we explore the current leading vendors of DDoS protection systems such as Akamai, which is responsible for serving 15 to 30 percent of the world wide web traffic, or the well-known company Cloudflare, that offers unmetered DDoS protection even for their free plans.**

***Index Terms*—Denial-of-service, distributed denial-of-service, distributed denial-of-service mitigation, DDOS, networks**

1. Introduction

The Internet is becoming increasingly important to society as billions of devices are now networked and more are being added every day. This increasing importance of accessibility means that there is also an ever greater incentive to disrupt it. One of the most common attacks on online services are Distributed Denial of Service (DDoS) attacks, which have become more frequent and more intense in the last two decades. Large attacks in Q1 2020 were again breaking records in peak bandwidth with Amazon reporting a 2.3 Terabits/s attack on their AWS servers [1]. Comparing this to the 5-6 Terabit/s average bandwidth of Frankfurt's internet exchange point DE-CIX in Q1 2020 reveals the size of such an attack [2].

A DDoS attack tries to overload a service with various methods with the goal of the service being unable to answer requests of legitimate users. A DDoS attack is a special kind of DoS (Denial of Service) attack, in which the source of the attack is distributed over multiple devices that cooperate to overwhelm the targeted service. The attack devices are often botnets, which are networks of compromised computers under the control of the attackers. Creating such a network is often accomplished by infiltrating computers through the usage of malware such as trojans and worms.

This stealing of computing and network resources already creates a great imbalance in expenses between attacking and defending side. That is an invitation for many attackers to attack their targeted web services in order to inflict financial damage as well as harming the public image. As defending against such sophisticated attacks that grow in size every year is no easy task, this has given a rise in popularity of DDoS protection provider to hide web services behind or in their large cloud networks [3].

In Section 2 the taxonomy of DDoS attacks is explained with basic examples. The three components of a DDoS defence system: attack detection, traffic classification and attack response are discussed in Section 3. Section 4 compares current leaders in the market of DDoS protection services and outlines the increasing adoption of such services. In Section 5 the paper is concluded and future work is mentioned.

2. Types of Attacks

DDoS attacks can be grouped into different categories and these diverse types of attacks call for different defence mechanisms. Furthermore, there are attacks called multi-vector attacks which try to combat this by combining several attack techniques. An unfortunate property of an online service is the fact that successfully attacking the weakest link in the network can stop the whole network. A strong DNS server will not help in the case the webserver itself is overloaded with dummy requests from the attacker [4]. While not all attacks can be categorized perfectly, there exist three basic types of attacks:

2.1. Volume-based Attack

This is the most common type of attack. The targeted network node is attacked with a sheer amount of dummy requests created by the botnet controlled by the attacker with the goal of depleting the available network bandwidth. This results in legitimate traffic being unable to pass through and the service is taken down.

Since volume-based attacks need large bitrates to be successful, a common attack is the DNS amplification attack. This attack abuses the fact, that DNS requests can receive large answers compared to the size of the request. This amplification can reach an amplification factor of 50+. Since DNS is UDP-based, the attacker uses the victims IP address as source address, which in turn will be targeted by a large amount of DNS packets as can be seen in figure 1.

doi: 10.2313/NET-NET-2022-07-1_01

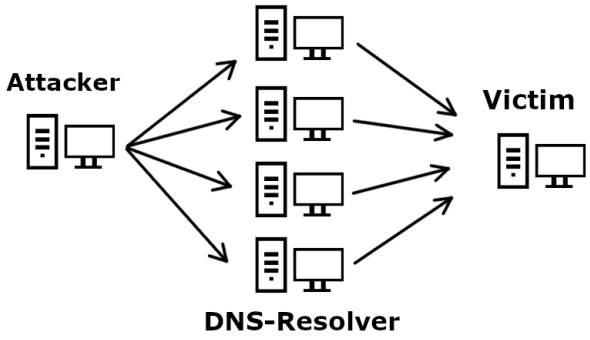


Figure 1: Congesting the network uplink of the victim with a DNS-based flood attack. (Reworked from Loukas et al. [5])

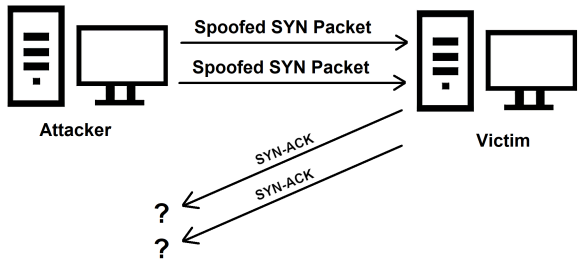


Figure 2: Depleting the resources of the victim with a TCP-SYN attack. (Reworked from [6])

2.2. Protocol-based Attack

These attacks mainly use vulnerabilities and shortcomings of protocols in OSI layers 3 and 4 to exhaust processing or memory resources of the target node instead of the network bandwidth. To be able to measure and compare the strength of these attacks, packets per seconds (pps) are usually used as metric.

A prominent example of this is a TCP SYN attack which exploits the way a TCP connection is established. The attackers send requests with active SYN flags to which the server responds with an SYN-ACK. Usually, the client would acknowledge this but in this case the attacker does not answer at all as it can be seen in Figure 2. Now the server has to wait for the ACK to timeout, wasting valuable memory space. To increase the difficulty of defending against this attack, the attacker will most likely also spoof the source IP address of the SYN packets.

2.3. Application Layer Attack

Attacks in this category are destructive compared to the small effort on the attacker side. They are also harder to detect as they often closely resemble real user behaviour.

An example for an application layer attack is the Slowloris attack. It abuses the HTTP protocol by sending incomplete HTTP GET requests without termination code and refreshes the connection just before the server would timeout the corresponding session. Over time this will occupy all connections the server is able to open at the same time and consequently service is unreachable for all users.

3. DDoS Mitigation Techniques

DDoS Mitigation can happen at various locations in the network. While we want to be as close to the source as possible to prevent the malicious network load from reaching large parts of the network, the distributed aspect of the attack makes this a challenging task. In a real-world scenario a service operator has the choice between three basic operating approaches. He can either run his own mitigation solution, outsource it to a DDoS Protection Service (DPS) provider or use a hybrid approach combining both solutions, each with its own benefits and downsides.

For most attacks the DPS should include the following three components [5]:

- **Detection:** First step in mitigating an attack is the simple detection of said attack. Attacks became more sophisticated and therefore distinguishing flash events from DDoS attacks has become harder. Detection can be grouped into anomaly-based and signature-based detection systems. While in anomaly-based detection the DPS has to first learn the normal user behaviour and later on detect an abnormal deviation to that, the signature-based detection tries to fit current observations into known patterns to detect attacks.
- **Classification:** As soon as an attack has been detected, the next step is to classify the incoming traffic into legitimate and invalid (created by the attacker) traffic.
- **Response:** After the malicious traffic has been marked the DPS needs to drop the invalid packets, preferable at the network edge to be less affected by the massively increased network traffic.

3.1. Detecting a DDoS Attack

The detection of the attack is the first step to be able to act on it. While it may sound like a simple comparison between normal traffic and the high volume traffic of an attack, there are also legitimate events that generate a high-volume of traffic. This could be an announcement, a product release or a news article linking to the specific service. In that case, dropping packets could heavily impact the companies behind the webservice either financially or in public reception. Furthermore, the attacks themselves are evolving and emerge in different shapes and sizes. Just by polling and comparing the traffic bandwidth alone it will be difficult to recognize an application-layer attack that does not rely on a huge attack bandwidth.

A common way to distinguish detection methods is to classify them either as anomaly-based or signature-based [3]:

Signature-based detection compares current network traffic with known attack behaviour, resulting in a high detection ratio and low false-positive rate. This is only true for known patterns and will be relatively ineffective for newly emerging attacks.

Anomaly-based detection will in most cases have a higher false positive rate, but also be effective in detecting new types of attacks. Anomaly-based detection divides

further into statistical analysis and machine learning based detection. In statistical analysis the system observes metrics such as packet arrival rate, packet type arrival rate and entropy of packet header fields. While it provides fast detection rates, without further inspection the false positive rate may be high. In learning-based systems, data mining techniques highlight previously unknown connections in the incoming traffic [3].

3.2. Classifying DDoS Traffic

A closely connected step is the classification of the incoming traffic, to be able to separate any legitimate traffic from the DDoS attack packets. As with detection, the classification works in a signature- or anomaly-based fashion and compares features to usual known traffic patterns. Features can be real-time gathered statistical features or even actively created by letting the user prove their legitimacy. This task is usually done by *dedicated validity tests*, which can be passive or active [5].

3.2.1. Passive Validity Tests.

Loyal clients: A very basic thought example is that a user that requests a news site every day, can be regarded as highly trustworthy even during times of an attack. This works even in cases when attackers spoof the source IP address of their packets, as it is unlikely, that they will randomly find a trusted IP address.

Time-to-live: Even though source addresses during an attack can and will often be invalid, the hop count of an IP packet will give insight into how far the packet travelled. This is done by Hop-Count filtering introduced by Jin et al. through comparing the TTL of the incoming packet with a table that stores known mappings between IP-addresses and their hop-counts [7]. It might not be a working traceback method but it can be compared to the apparent source IP address and give hints about the legitimacy of the packet.

3.2.2. Active Validity Tests.

Active validity tests in comparison are in direct communication with the user of the incoming request and challenge it to prove its legitimacy. This has become an important step since sophisticated attacks started to imitate a natural increase in bandwidth similar to external events. Therefore, classifying this kind of traffic has become harder. Active tests work on the premise that legitimately increased traffic patterns will be created by humans instead of automated programs. Famously used for this task are Reverse Turing tests, such as CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart).

While CAPTCHAs come in different types, all of them are based on challenges trivially solvable by a human but hard to solve by a computer. Challenges range from tasks like reading obscured digits and letters to classifying a group of images. In 2013 Google also included a behaviour-based analysis of the browser interactions as a filter in its CAPTCHA service called reCAPTCHA. This improves usability for deemed low-risk users as they are not tasked to solve time-consuming challenges. In case the fingerprinting does not rate the user as credible, another verification with classic challenges is added successively.

3.3. Responding to an DDoS Attack

The DDOS attack responses are typically classified depending on their location in the network. They are classified in source-based, network-based and destination-based techniques each with its own advantages and disadvantages as shown by Dietzel et al. [8]. Due to the distributed aspect of DDoS attacks, the easiest position to detect an attack is directly at the target (destination-based) but the mitigation is less effective. Mitigating an attack close the source would be ideal but is difficult to realise in practice as attacks can be launched from anywhere. The following paragraphs outline a selection of techniques used in practice:

A destination-based mitigation technique is dropping the packets that have been marked high-risk by the classification component. As mentioned in the last section, these filters work time-based, history-based or hop-count-based.

Especially with volume-based attacks, filtering alone will not alleviate the pressure on the network resources. Even if the heuristics would allow for a good classification, the number of incoming packets would still overload the target network. For this reason and as an additional mitigation step, mechanisms such as adaptive rate limiting by Ioannidis et al. [9] or IP traceback by Adler et al. [10] are proposed.

Adaptive Rate Limiting is based on the concept of aggregates which are subsets of the traffic that share a common property. These properties include the packet destination, the type of packet and packets with a bad checksum. If an aggregate responsible for a significant portion of the traffic is found, the aggregate is propagated to upstream routers to rate-limit the malicious traffic. Traffic adhering to the rate-limit will be allowed while other traffic will be dropped to as mentioned by Zargar et al. [11].

IP Traceback mechanisms try to find the true sources of the forged IP packets. Since IP routing is stateless and routers usually only know where to forward the incoming packet, the routers have to support the traceback method to be able to contribute to IP traceback mechanisms. The main categories of traceback techniques are packet marking and link testing. In packet marking the routers add their identification to the packet probabilistically in order to enable the victim to identify the path of malicious traffic after receiving enough packets. In link testing the routers closest to the victim get iteratively tested until the source of the attacker's traffic can be reached [11]. The effectiveness of IP traceback in practice is limited, since the traceback mechanism would need to be deployed with minimal cost in time and storage, low false positive rate and while respecting privacy of the inspected packets [5]. Attackers can also forge their own marked packets and therefore disturb the traceback mechanism.

Increasing the Attack Surface may sound a little contra-productive as reducing it is an important step in many cyber security related topics. The nature of DDoS attacks however concentrate on a single point which makes mitigating the attack almost impossible if the attack surface is reduced to a certain point [3]. Therefore increasing

doi: 10.2313/NET-NET-2022-07-1_01

the attack surface is a common strategy to help mitigate an DDoS attack or at least soften the impact on the targeted network. In practice this is usually achieved by using cloud providers that hide the user either in or behind their large networks.

An effective last resort response to a DDoS attack is the so-called *blackholing* of the traffic flowing to the victim. Blackholing is defined as dropping the traffic at the routing level, which can be implemented at almost any router with no additional performance impact. The victim autonomous system announces the prefix to blackhole to its upstream network via BGP (Border Gateway Protocol) which nowadays means the blackholing will atleast happen at one of the supporting IXPs (Internet Exchange Point) [8]. Blackholing is usually the last resort since the announced prefixes will by unreachable by both the attackers and the legitimate users, but it will reduce collateral damage to neighbouring devices and networks.

4. DDoS Protection Service Provider

Many web services today rent a cloud service to allow them to cost-effectively scale their operation. Many of the cloud service provider also offer their cloud as a DDoS Protection Service. The traffic will be sent through the cloud where the traffic will be cleaned from malicious packets in so called scrubbing centers. The clean traffic will be rerouted to the webserver responding to the requests of valid users.

DDoS protection via a cloud service can either be always-on, on-demand or a hybrid version that combines both. The most common always-on solution is the usage of a CDN (Content Delivery Network), which distributes the content over many cache servers around the world to be geographically close to the end user. CDNs are not only used for QoS (Quality of Service) objectives, they can also be used as DDoS protection as the distribution of content reduces the effects of an attack [3].

In case an on-demand strategy is desired, a reactive plan that reroutes the traffic only in the case of an attack to the cloud is appropriate. The cloud then scrubs the incoming packets and sends back the clean traffic to the webserver. The routing of the traffic can either be done by making a change in the DNS record of the victim or by a BGP advertisement change.

4.1. DPS Provider Overview

In Forrester Research' 2021 report of DDoS Protection Service provider 11 significant vendors are mentioned and compared against each other [12]. This paper focuses on the 4 leaders in the market according to Forrester Research which are Akamai, Cloudflare, Imperva and Radware. While the vendors keep their filter technology and scrubbing techniques proprietary, the mechanisms are not significantly different compare to on-premise detection [4].

Akamai is one of the largest cloud service providers, with a network of over 300,000 servers in 135 countries serving between 15% and 30% of the web traffic. This large network size accumulates to a network capacity of more than 175Tbps. Akamai is targeted towards enterprise customers, as it has a minimum contract of 12 months and does not reveal pricing information without requesting a quote [12].

Cloudflare is another big CDN provider with a strong focus on DDoS mitigation. In comparison to the enterprise focused Akamai, Cloudflare offers start with a free plan containing the option of adding additional features via their Pro and Enterprise plans priced $20 and $200 per domain respectively. Their basic volumetric DDoS protection is already included in the free plan, which is also unmetered. Defence mechanisms against layer 7 attacks have to be purchased additionally as a package [13].

Radware is an Israelian company that offers application delivery and several cybersecurity products. In contrast to Akamai and Cloudflare, Radware is one of the oldest and largest vendor for on-premise DPS devices, but they have been transitioning towards cloud-based and hybrid approaches in the last years. Due to their long presence in the industry they have a deep understanding of DDoS attacks. Therefore, they are especially suited for difficult attack cases [12].

Imperva is another security specialists that offers cloud-based DDoS protection services. Imperva advertised it's large network size in the last years but according to [12], most of their competitors have caught up and even surpassed the capability of Imperva's network. The capability to deflect even the largest attacks can currently still be found at all large cloud providers.

4.2. DPS Adoption

The increasingly large DDoS attacks every year also increase the pressure on web service provider to employ a cloud-based DPS to be able to mitigate them. Jonker et al. have proposed a methodology to check domain names for active traffic diversion to a cloud-based DPS and used it to analyse all .com, .net and .org TLDs containing over 50% of the names in the global namespace with daily snapshots over 1.5 years between March 2015 and September 2016. While the amount of domains in that namespace grew 9% from 140M to 152M domains, the number of domains protected by the top 9 leading DPS provider grew 24% to a total of about 9M domains. For 6 months they have also been monitoring the Alexa Top 1M list as well as the .nl TLD, whose DPS usage grew 12% and 11% during that time respectively [14]. This shows clearly the increased interest in the services of DPS providers.

5. Conclusion and Future Work

In this paper we provided an overview of basic DDoS attack types and general mitigation strategies. This was followed by an overview of leading vendors in the DPS market. Thereafter, the ongoing trend towards cloud-only or hybrid-based DDoS Protection Services was outlined. They are becoming the most popular remaining option to have a chance against these Terabits per second large volumetric attacks, which have occurred increasingly often in the last few months and years. Future challenges and opportunities lie in the field of SDN (software-defined networking) and the usage of machine learning based detecting and filtering of malicious traffic.

doi: 10.2313/NET-NET-2022-07-1_01

References

[1] "AWS Shield - Threat Landscape Report," May 2020. [Online]. Available: https://aws.amazon.com/de/blogs/security/aws-shield-threat-landscape-report-now-available/

[2] "DE-CIX Traffic Statistics." [Online]. Available: https://www.de-cix.net/en/locations/frankfurt/statistics

[3] I. Ozcelik and R. Brooks, *Distributed Denial of Service Attacks: Real-world Detection and Mitigation*, 05 2020.

[4] E. Chou, R. Groves, and a. O. M. C. Safari, *Distributed Denial of Service (DDoS): Practical Detection and Defense*. O'Reilly Media, 2018. [Online]. Available: https://books.google.at/books?id=G19PwAEACAAJ

[5] G. Loukas and G. Öke, "Protection Against Denial of Service Attacks," *Comput. J.*, vol. 53, no. 7, p. 1020–1037, Sep. 2010. [Online]. Available: https://doi.org/10.1093/comjnl/bxp078

[6] "SYN-Flood-Attack, Cloudflare Learning." [Online]. Available: https://www.cloudflare.com/de-de/learning/ddos/syn-flood-ddos-attack/

[7] C. Jin, H. Wang, and K. G. Shin, "Hop-Count Filtering: An Effective Defense against Spoofed DDoS Traffic," ser. CCS '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 30–41. [Online]. Available: https://doi.org/10.1145/948109.948116

[8] C. Dietzel, A. Feldmann, and T. King, "Blackholing at IXPs: On the Effectiveness of DDoS Mitigation in the Wild," in *International Conference on Passive and Active Network Measurement*. Springer, 2016, pp. 319–332.

[9] J. Ioannidis and S. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," 03 2002.

[10] M. Adler, "Tradeoffs in Probabilistic Packet Marking for IP Traceback," *Journal of the ACM (JACM)*, vol. 52, no. 2, pp. 217–244, 2005.

[11] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[12] D. Holmes, "The Forrester Wave: DDoS Mitigation Solutions, Q1 2021," *Forrester Research*, March 2021.

[13] "Cloudflare Pricing and Plans." [Online]. Available: https://www.cloudflare.com/plans/#overview

[14] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras, "Measuring the Adoption of DDoS Protection Services," ser. IMC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 279–285. [Online]. Available: https://doi.org/10.1145/2987443.2987487

Comparison of Different QUIC Implementations

Salim Hertelli, Benedikt Jaeger*, Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: hertelli@in.tum.de, jaeger@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—QUIC is an encrypted and multiplexed transport protocol developed by Google and deployed on their servers in 2012. QUIC aims to replace the commonly used TCP/TLS stack. It was standardized on the 27th of May 2021 in the RFC 9000 [1], which defines the core and specifications of the protocol. QUIC is designed to be implemented in the user space. This allows different implementations to exist in multiple programming languages and with different features. This paper aims to give an overview of existing implementations and to compare them based on different metrics, such as the used programming language, supported versions of QUIC, handshake encryption method, and used congestion control algorithm.**

***Index Terms*—quic, transport protocol, http/3, quic implementaions**

1. Introduction

QUIC is a connection-oriented, encrypted, and multiplexed transport protocol built on UDP as shown in Figure 1. It was developed and deployed by Google in 2012 as a replacement for the traditionally used TCP/TLS stack, commonly used in the HTTPS stack. QUIC aims

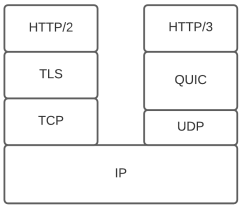


Figure 1: QUIC stack vs TCP/TLS stack (adapted from [2]).

to improve on the pre-existing protocols by enhancing security and reducing latency. It was standardized by the IETF[1] with the release of RFC 9000 in May 2021, which was complemented by three more documents, namely RFC 8999, 9001, and 9002.

As part of its design goal, QUIC is implemented in the user space. This property of QUIC leads to faster development and deployment cycles as it avoids the long process of pushing system-wide updates [2]. This also allowes for different congestion avoidance algorithms to be dynamically used, which makes it easier to perform experimentation using various congestion control algorithms, fix bugs, and deploy changes. Since then multiple implementations have emerged in different programming languages and paradigms, including functional programming languages such as Haskell.

In this paper, we list some of the available QUIC implementations. Differences between them will be analyzed based on various metrics and criteria, such as the status of the projects, how well-maintained they are, etc. Section 2 introduces some of the specifications of QUIC. In Section 3 we discuss the methodology we used to collect data about different projects. We then display the results in Section 4. In Section 5 we highlight other research that was conducted on QUIC. We then conclude the paper in Section 6.

2. Background

QUIC was started as an experimental protocol by Google back in 2012 to replace the existing TCP/TLS stack used for HTTP. By being developed to be deployed in the user space and not in the kernel of operating systems, it allows for faster development cycles. It does also allow for critical updates to be pushed and applied faster and gives developers much more room to experiment with new features and improvements [2].

In 2015 a draft of QUIC was submitted and a working group was created at the IETF with the goal of standardizing QUIC. This standardization came on May 27th, 2021 with the release of RFC 9000. It specifies the core of the QUIC protocol and serves as a certification for QUIC's reliability [1].

As detailed in the RFC 9000 [1], QUIC supports flow streams and network path migration among other features. In addition, QUIC allows multiple streams to be multiplexed, which prevents head-of-line blocking. This leads to a reduced latency overall compared to TCP, as the costs of using multiple TCP connections can be mitigated. Handshake delays are improved as well by removing unnecessary round trips. This allows exchanges to occur as early as possible or even immediately, leading to 0-RTT handshakes in some cases [2]. QUIC also improves congestion control [2]. RFC 9000 does not specify a particular algorithm to be used, this allows researchers to experiment and improve on it.

QUIC fully encrypts and authenticates handshakes. This holds for almost all the handshakes except some of the early ones. Encryption covers the majority of a QUIC package. The unencrypted parts are needed for

1. Internet Engineering Task Force

doi: 10.2313/NET-NET-2022-07-1_02

routing purposes such as connection ID and version number among others. In addition, the authentication process ensures that packets that have been tampered with can be discovered, which leads to a connection failure [2].

As mentioned previously, QUIC has many different implementations. Different implementations may support different versions of QUIC depending on the starting time of the project. At the time of writing, as it is just a couple of months after the standardization of QUIC, not all implementations support QUIC version 1. It is however prone to change with time.

3. Methodology

In this section, we discuss the process, tools, and methodology used to collect data about different QUIC implementations. As stated earlier, there exists various QUIC projects in many programming languages [3]. The choice of which implementation to select for this paper was then done based on different criteria including project age, the status of the implementation, and whether the project is being backed up by a big company. Chromium-quic was the first selected implementation due to the fact that Google was the company to start the development of QUIC and because it powers one of the most prominent browsers, namely Google Chrome. The next selected implementations are Quiche, Mvfst, and Msquic which are backed by Cloudflare, Facebook, and Microsoft respectively. Quic-go was then selected based on the debuting time of the project, which started in early 2016. The last two implementations are Aioquic and Haskell-quic, developed respectively in Python and Haskell. This was done to show diversity in the used programming languages for QUIC projects.

The next step was to collect data about the pre-selected projects. Project-specific properties were collected in an automated process. Many of the selected projects are hosted on GitHub. This allowed the usage of the GitHub API, which can be queried using scripts to collect the project creation date, number of pull requests, and the number of commits. It does also provide information about the main programming language of a project. However, this may lack a certain level of accuracy, considering the large number of commented lines in big coding projects. To collect exact information about the utilized programming languages, Cloc was used [4]. Cloc is an open-source software written in Perl used to count lines of codes in a directory. It does then present an overview of the results, including, but not limited to the used programming languages based on file extensions and how many lines of code each file contains. Moreover, it separates the commented lines from the actual code lines as well.

4. Evaluation

In this section, we display the collected results about the different QUIC implementations. The collected data is summarized in TABLE 1 and TABLE 2 and will be further discussed in this section.

4.1. Chromium-quic "Quiche"

Chromium-quic is a QUIC implementation and part of the Chromium projects, which are open-source projects developed by Google. The project is called Quiche as an acronym for "QUIC, HTTP/2, Etc". It is a production-ready implementation, written exclusively in C and C++. It powers parts of the "Google" search engine and the "YouTube" video playing service. The project is hosted on Google's servers [5] as well as on GitHub and is kept in sync. The code is well maintained and documented with an extensive wiki and many supporting documents, such as the RFCs. The project did go through a lot of QUIC versions, starting from Q403 until draft-29. It supports CUBIC for congestion control and uses QUIC-Crypto as well as TLS to encrypt the packets being transmitted [3].

4.2. Cloudflare "Quiche"

Quiche[2] is another implementation of the QUIC protocol written in Rust and hosted on GitHub [6]. It was developed by Cloudflare, a web security and infrastructure company, in order to enable HTTP/3 support on their servers. Documentation for the project does provide a guide on how to build and configure Quiche to receive, send and handle packages. The wiki also has a listing of the used structures, enum, and functions used in the implementation as well as short descriptions to help developers understand their functionality.

Quiche supports the usage of two different congestion control algorithms namely Reno and CUBIC as well as a high-level API in order to configure the used algorithm. CUBIC is deployed on Cloudflare's production environment. Later came the introduction of HyStart++ to improve congestion control. HyStart++ is a modification of the slow start phase in congestion control algorithms, which tries to improve the performance by reducing packet loss and preventing the overshooting of the ideal sending rate. This is done by introducing the Limited Slow Start phase (LSS). Before reaching the congestion threshold in the Slow Start phase, the congestion control algorithm switches to LSS. During the LSS phase, the congestion window grows slower than in the congestion avoidance phase. Upon reaching the congestion threshold, the algorithm switches then to the congestion avoidance phase [7].

4.3. Mvfst

Mvfst is an implementation developed by Facebook. It is mainly written in C/C++ and hosted on GitHub [8]. More than 75 percent of all the network traffic of Facebook is happening on QUIC and HTTP/3. This includes their social networking websites Facebook and Instagram. Mvfst makes use of Facebook's own TLS 1.3 implementation "Fizz" to ensure the security of packet exchanges. The implementation comes with a wiki that explains how to build, run and test Mvfst. It does also provide samples for both client and server side. The wiki does not mention, which congestion control algorithm Mvfast uses. However, the header files and the implementation for both NewReno and CUBIC are present in the source code [9].

Moreover, Facebook does experiment with artificial intelligence based congestion control algorithms. Mvfst-rl

2. Not to be confused with the Chromium QUIC implementation which is also called Quiche.

doi: 10.2313/NET-NET-2022-07-1_02

TABLE 1: Listing of different QUIC implementations and metrics about the status of the projects (September 2021)

Project	Language	License	Creation date	LoC	#pull requests	average/day	#commits	average/day
Quic-go	Go	MIT	06/04/2016	61119 Go	1843	0.92/day	678	1.82/day
Chromium-quic "Quiche"	C / C++	BSD-3-Clause	-	228 C 211574 C++ 40881 C/C++ headers	-	-/day	-	-/day
Cloudflare "Quiche"	Rust / C	BSD-2-Clause	29/09/2018	36920 Rust 1300 C	715	0.65/day	294	0.79/day
Aioquic	Python	BSD-3-Clause	05/02/2019	17337 Python	115	0.11/day	87	0.20/day
Mvfst	C / C++ Python / Rust	MIT	10/04/2018	75760 C++ 15927 C/C++ headers 6537 Python 1150 Rust	124	0.09/day	1,317	3.58/day
Msquic	C / C++	MIT	26/10/2019	56291 C 27155 C++ 39296 C/C++ headers	1,529	2.18/day	756	2.07/day
Haskell-quic	Haskell / C	BSD-3-Clause	11/01/2019	9741 Haskell 5503 C	10	0.01/day	481	1.30/day

TABLE 2: Different QUIC implementations and their used versions, handshake encryption methods and congestion control algorithms (September 2021)

Project	Versions	Roles	Handshake	Congestion Control
Quic-go	as the current draft	library, client, server	TLS 1.3 RFC	NewReno, CUBIC
Chromium-quic "Quiche"	Q043, Q046, Q050, T050 T051, draft-27, draft-29	library, client, server	QUIC Crypto, TLS	CUBIC
Cloudflare "Quiche"	draft-27, draft-28, draft-29	library, client, server	TLSv1.3 (RFC8446)	NewReno CUBIC + HyStart++
Aioquic	draft-29, version 1	library, client, server	TLS 1.3	NewReno
Mvfst	draft-29	library, client, server	TLS 1.3	NewReno, CUBIC Mvfst-rl (experimental)
Msquic	draft-29, version 1	client, server	TLS 1.3 RFC	CUBIC
Haskell-quic	draft-29	library, client, server	TLS 1.3	NewReno

[10] is a framework that uses asynchronous reinforcement learning training in order to improve congestion control in QUIC and is built on their own implementation. However, it is still an experimental feature and is yet not ready to be deployed in a production environment.

4.4. Msquic

Msquic is an IETF QUIC implementation written in C and C++. The project was started by Microsoft and is hosted on GitHub [11]. Currently, it supports both Draft-29 and version 1 of QUIC. Msquic uses TLS 1.3 to encrypt and authenticate all handshakes and packets. As for congestion control, it supports CUBIC. The project comes with extensive documentation to build, test and deploy Msquic. Microsoft does also provide daily benchmark results including single connection upload and download rates and the average number of requests completed per second. Msquic is a production-ready implementation. It powers Microsoft's HTTP/3 stack and is deployed in other products to handle QUIC connections [12].

4.5. Quic-go

Quic-go is a QUIC implementation in the Go programming language and is hosted on GitHub [13]. Currently, it does implement the IETF draft-29. The documentation in their GitHub repository indicates however that the support for draft-29 will eventually be replaced by a more recent standard. Quic-go does support both NewReno and CUBIC for congestion control and uses TLS 1.3 to encrypt handshakes and packages. The implementation comes with examples for client and server instances as well as instructions on how to run tests.

4.6. Aioquic

Aioquic is an IETF QUIC implementation in Python. It is an open-source project hosted on GitHub [14]. The implementation is built on Asyncio, which is a standard Python framework for asynchronous I/O. Aioquic is implemented to be conforming with the RFC 9000. It does support a minimal TLS 1.3 implementation for packages and handshake encryption and it uses NewReno for congestion control, as recommended by the RFC 9000, which features pseudocode for NewReno. Aioquic's wiki explains how to test the implementations on different operating systems including Windows, Linux, and MacOS. There are also different examples, which can be used to test different QUIC use cases.

4.7. Haskell-Quic

Haskell-quic is an implementation of IETF QUIC in Haskell and is hosted on GitHub [15]. The main difference of this implementation compared to the previous ones

is that Haskell is a functional programming language. This makes supporting new features harder, as they are generally described in an imperative style pseudocode. The pseudocode needs then to be transfered to functional style in order to implement it in a functional language like Haskell. Kazuhiko Yamamoto, the main developer behind Haskell-quic, claims that this transformation is not a simple task to perform [16].

Haskell-quic is based on Haskell's lightweight threads and uses TLS to secure handshakes. The author does maintain a blog [17] to track the development of haskell-quic where he discusses important milestones and future features to be implemented. Haskell-quic uses NewReno for congestion control as recommended by the RFC 9000.

5. Related work

The IETF QUIC Working Group maintains a listing [3] in order to track known implementations. The listing also keeps track of the used programming languages, versions that are implemented, handshakes encryption methods, and, if available, some links to public servers that use or allow experimentation to be conducted with the corresponding implementation.

Marx et al. [18] compare different QUIC implementations and discuss their behavioral heterogeneity. They also discuss more behavioral aspects than those mentioned in this paper, including multiplexing scheduling and the 0 RTT approach used. In addition, they apply different methodologies by using the qvis and qlog tools.

Interoperability tests play an important role in internet protocols development, including QUIC. Interoperability tests check whether different independently developed implementations interact with each other as expected. Marten Seemann and Jana Lyengar describe it as a crucial tool to expose weakness and ambiguities in the specifications of QUIC. Moreover, the IETF recommends testing to be a part of the development process [19]. One interoperability testing framework is QUIC Interop Runner or QIR. QIR performs different tests between all pairs of their listed implementations. This includes testing the server as well as the client functionalities on different scenarios such as handshake loss and version negotiation among others [20].

6. Conclusion

In this paper, we discussed QUIC, a transport protocol intended as a replacement for the TCP/TLS stack and implemented in the user space. We have shown 7 different QUIC implementations and presented different properties ranging from the status of the project, documentation, supported QUIC versions, and used congestion control algorithms. We utilized an almost automated process to collect data about various projects using different APIs and project analyzing software.

Although all the mentioned implementations follow the IETF specifications, our results show a large diversity in the used programming languages and supported features. In particular, the used congestion control algorithms differ from one project to another as no particular algorithm was specified by the IETF. However, we can notice that CUBIC and NewReno are almost always used in production. Supported versions of QUIC are also not the same across different implementations. This can however be linked to the age of the project. As the QUIC standard was published just a few months before the time of writing of this paper, not all implementations are currently supporting version 1 of QUIC. However, it is already supported by some of them, including Aioquic and Msquic.

QUIC is still relatively new compared to TLS. However, it is remarkable to see that some implementations are already production-ready and being used by different mainstream services that are accessed by millions daily. This protocol as well as its development process will play an important role in defining norms for creating new internet protocols in the future as QUIC is getting more popular and is beginning to be universally adopted.

References

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

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. B. Krasic, C. Shi, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. C. Dorfman, J. Roskind, J. Kulik, P. G. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, and W.-T. Chang, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," 2017.

[3] IETF QUIC Working Group, "QUIC Implementations," available at https://github.com/quicwg/base-drafts/wiki/Implementations, [Online. accessed September-2021].

[4] AlDanial, "Cloc," available at https://github.com/AlDanial/cloc, [Online. accessed September-2021].

[5] Google, "Quiche," available at https://quiche.googlesource.com/quiche/, [Online. accessed September-2021].

[6] Cloudflare, "Quiche," available at https://github.com/cloudflare/quiche, [Online. accessed September-2021].

[7] J. Choi, "CUBIC and HyStart++ Support in quiche," available at https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche/, [Online. posted on 20-August-2020].

[8] Facebook, "mvfst," available at https://github.com/facebookincubator/mvfst, [Online. accessed September-2021].

[9] F. Engineering, "How Facebook is bringing QUIC to billions," available at https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/, [Online. posted on 21-October-2020].

[10] Facebook, "mvfst-rl," available at https://github.com/facebookresearch/mvfst-rl, [Online. accessed September-2021].

[11] Microsoft, "msquic," available at https://github.com/microsoft/msquic, [Online. accessed September-2021].

[12] D. Havey, "MsQuic is Open Source," available at https://techcommunity.microsoft.com/t5/networking-blog/msquic-is-open-source/ba-p/1345441, [Online. posted on 28-April-2020].

[13] L. Clemente, "Quic-go," available at https://github.com/lucas-clemente/quic-go, [Online. accessed September-2021].

[14] "Aioquic," available at https://github.com/aiortc/aioquic, [Online. accessed September-2021].

[15] K. Yamamoto, "Haskell-quic," available at https://github.com/kazu-yamamoto/quic, [Online. accessed September-2021].

[16] ——, "Developing QUIC Loss Detection and Congestion Control in Haskell," available at https://kazu-yamamoto.hatenablog.jp/entry/2020/09/15/121613, [Online. posted on 15-September-2020].

[17] ——, "The Current Plan for Haskell QUIC," available at https://kazu-yamamoto.hatenablog.jp/entry/2020/10/23/141648, [Online. posted on 21-October-2020].

[18] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity," 08 2020.

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

[20] M. Seemann, "QUIC Interop Runner," available at https://github.com/marten-seemann/quic-interop-runner, [Online. accessed November-2021].

doi: 10.2313/NET-NET-2022-07-1_02

Optimizations for Secure Multiparty Computation Protocols

Thilo Linke, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: thilo.linke@tum.de, christopher.harth-kitzerow@outlook.de

***Abstract*—The Beaver-Micali-Rogaway protocol describes a method for securely computing functions with any number of participating parties, that builds on the principles of Yao's Garbled Circuits protocol for two participants. Its key advantage over similar protocols is that it only requires a fixed constant amount of communication rounds to build the garbled circuits. Moreover, it is possible to apply the FreeXOR optimization technique to the protocol in order to simplify the evaluation of XOR gates of the garbled circuits and thereby improve overall runtime.**

***Index Terms*—secure multiparty computation, bmr protocol, freexor optimization technique**

1. Introduction

A simple practical scenario that requires the usage of a Secure Multiparty Computation (SMC) protocol is to carry out a private vote. The concrete purpose of SMC is to provide protocols that keep computation input data from participants private, while not requiring any trusted third parties [1].

Yao's Garbled Circuits protocol (YGC) for two parties, introduced in 1983 by A. C. Yao [2], started the long evolution of SMC protocols [1]. The GMW protocol from O. Goldreich et al. [3] was one of the first to enable computations with any number of participants.

This paper presents the Beaver-Micali-Rogaway protocol (BMR), introduced in 1990 by Donald Beaver, Silvio Micali and Phillip Rogaway, which generalizes YGC's concepts in order to enable any number of participants [4], [1]. Its constant number of required communication rounds for building its computation structure makes it especially attractive for scenarios with high network latency, like computation over the internet [5].

After providing an overview of the original BMR protocol, we describe an optimization introduced by A. Ben-Efraim et al. in 2016 [5], who apply the FreeXOR technique to evaluate XOR gates of garbled circuits more efficiently.

2. The BMR Secure Multiparty Computation Protocol

The BMR protocol adapts the garbled circuit concept from YGC, which cryptographically guarantees that input values are kept secret. Before actually describing the protocol, Section 2.1 serves as an introduction to this concept. The following description is based on the circuit definition from the original version of the BMR protocol from [4]. We use symbols similar to those from the BMR protocol adaptation from [5].

2.1. Garbled Circuits

A garbled circuit consists of wires, signals and gates. These building blocks can be used to construct any arbitrary computable function, like in an ordinary Boolean circuit.

2.1.1. Wires. The wires can carry one of two signals and connect the gates. Each party initially holds some data for the circuit input wires and the combined input from all parties is needed to execute the circuit. The values obtainable from the circuit output wires are the result of the computation.

2.1.2. Signals. Garbled circuits encrypt the Boolean signal values to hide them from the participating parties and ensure input secrecy.

Like in the YGC protocol, wire ω does not carry signals 0 or 1, but the secret random binary strings $k_{\omega,0}$ or $k_{\omega,1}$, which get collaboratively generated by the parties. The key modification of Yao's two-party method is that each party i of the n computing parties possesses its own private share of the strings in form of substrings $k^i_{\omega,0}$ and $k^i_{\omega,1}$ of the length of the cryptographic security parameter κ [1]. The two private signals for each wire therefore are

$$k_{\omega,\tau} = k^1_{\omega,\tau} \cdots k^n_{\omega,\tau} \text{ for } \tau \in \{0,1\}. \tag{1}$$

During circuit creation, each party additionally generates a secret share λ^i_ω of a random *permutation bit* λ_ω for each wire ω. The real meaning of a signal string $k_{\omega,\tau}$ of the circuit is then defined as Boolean value

$$\lambda_\omega \oplus \tau = \lambda^1_\omega \oplus \cdots \oplus \lambda^n_\omega \oplus \tau \tag{2}$$

and because nobody knows the value of λ_ω, the hidden Boolean signals are effectively concealed. This is the mechanism that actually enables input privacy. [5]

2.1.3. Gates. A gate g with left and right input wires α and β, as well as output wire γ calculates an arbitrary Boolean function on its inputs. Since the input and output signals are random strings that conceal their underlying values, the gates have to work with a specific mechanism to enable output computation.

Each gate g holds a table of four $n \cdot \kappa$ bit long *gate label* strings $X^{a,b}_g$ for each possible combination of inputs

doi: 10.2313/NET-NET-2022-07-1_03

$k_{\alpha,a}$ and $k_{\beta,b}$ [4]. Informally, the labels are associated with the signals via a truth table. More formally:

$$((\alpha \text{ carries } k_{\alpha,a}) \wedge (\beta \text{ carries } k_{\beta,b})) \longleftrightarrow X_g^{a,b} \quad (3)$$

The core of the gate label creation process is to seed a pseudorandom generator with the gate input signal strings and then to mask the gate output signal strings with the result. The encrypted output signals are then used as gate labels. Details on how this can be achieved and how circuit evaluators use the labels to compute the output of a gate can be found in the following section.

2.2. The Protocol

Overall, the BMR protocol computation process is commonly divided into two phases. The first phase utilizes SMC between the n participants to generate the garbled circuit and inputs, while in the second phase, each party executes the built circuit on their own to obtain the result.

This section explains the structure and required steps of the original protocol from [4] in order to provide a basic foundation. There exist many subsequent descriptions, like from [5], [6] or [7], that additionally apply various modifications. We mention two of those enhancements in Section 2.3.

2.2.1. Phase 1. The protocol starts by constructing the garbled circuit and its garbled inputs. Any secure protocol like BGW or GMW can be used for steps that require SMC [1], [5], [6].

Signal Creation. *Secret sharing* is used to generate random bit strings. Essentially, each party i privately generates random bit strings s^i that are called shares. The actual value is then defined as

$$s = \bigoplus_{i=1}^{n} s^i. \quad (4)$$

This means that the resulting value remains secret, unless someone possesses all shares at once. [4]
The two steps to compute the signals are:

1) The parties generate the permutation bits λ_ω for each wire ω by creating the private λ_ω^i shares [4].
2) They additionally need to create the secret random signal strings $k_{\omega,\tau}$ for $\tau \in \{0,1\}$ [4]. After this step, each party holds private substrings $k_{\omega,\tau}^i$ for each wire ω.

Label Creation. As we mentioned in Section 2.1.3, each gate will hold a table of four strings, called gate labels. The original BMR design additionally associates all wires of the garbled circuit with public labels. Because the creation of the labels can be accomplished in parallel and the required communication is independent of the size of the circuit, it is very efficient [1].

To generate the labels, party i locally uses a pseudorandom generator G, which takes each of their previously obtained private signal substrings of length κ for wire ω as input and transforms them to pseudorandom strings of length $\kappa + 2 \cdot n \cdot \kappa$. To be precise,

$$G(k_{\omega,\tau}^i) = x_{\omega,\tau}^i y_{\omega,\tau}^i z_{\omega,\tau}^i, \quad (5)$$

where $|x_{\omega,\tau}^i| = \kappa$, $|y_{\omega,\tau}^i| = n \cdot \kappa$ and $|z_{\omega,\tau}^i| = n \cdot \kappa$. [4]

Each party has to prove via zero-knowledge-proofs to the other parties that they truthfully calculated these strings, as a measure to rule out malicious intent [4].

The produced strings are processed as follows:

1) The $x_{\omega,\tau}^i$ strings for $\tau \in \{0,1\}$ are used to form public *wire labels* $x_{\omega,\tau} = x_{\omega,\tau}^1 \cdots x_{\omega,\tau}^n$ [4]. If a circuit evaluator obtains signal $k_{\omega,a}$ for a wire ω during the second phase, they can use G to calculate the same wire label that the signal produced previously in phase one. The knowledge about which of the two publicly known wire labels they obtain from this allows them to choose the correct gate label to proceed (see Section 2.2.2).
2) The $y_{\omega,\tau}^i$ and $z_{\omega,\tau}^i$ strings for $\tau \in \{0,1\}$ are used for collaborative gate label creation and remain private [4]. Each of these labels encrypts one output signal string of a gate. Because G used the input signals as seeds, they are also the keys to decrypt the output signals. The association between input signals and gate labels is shown in (3).
   For example, if a circuit evaluator holds signals $k_{\alpha,0\oplus\lambda_\alpha}$ and $k_{\beta,1\oplus\lambda_\beta}$ for left and right input wires α and β at AND gate g, the associated gate label for the signals should encrypt signal $k_{\gamma,0\oplus\lambda_\gamma}$ for output wire γ. The following equations, adapted from [4], that get securely and collaboratively evaluated by the parties using SMC, ensure this:

$$\begin{aligned}
X_g^{0,0} &= \bigoplus_{i=1}^{n} (y_{\alpha,0}^i \oplus y_{\beta,0}^i) \oplus k_{\gamma, f_g(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma},\\
X_g^{0,1} &= \bigoplus_{i=1}^{n} (z_{\alpha,0}^i \oplus y_{\beta,1}^i) \oplus k_{\gamma, f_g(\lambda_\alpha,\overline{\lambda_\beta})\oplus\lambda_\gamma},\\
X_g^{1,0} &= \bigoplus_{i=1}^{n} (y_{\alpha,1}^i \oplus z_{\beta,0}^i) \oplus k_{\gamma, f_g(\overline{\lambda_\alpha},\lambda_\beta)\oplus\lambda_\gamma},\\
X_g^{1,1} &= \bigoplus_{i=1}^{n} (z_{\alpha,1}^i \oplus z_{\beta,1}^i) \oplus k_{\gamma, f_g(\overline{\lambda_\alpha},\overline{\lambda_\beta})\oplus\lambda_\gamma},
\end{aligned}$$

   where $f_g(\cdot,\cdot)$ is the Boolean gate function, which would be AND in the previous example. Section 2.2.2 explains how the masked output signal can be obtained from a gate label, if the gate input is known to an evaluator.

Garbled Input Creation. By using SMC, the parties decide which of the two signals for each input wire ω gets chosen as input for the circuit. To do this, the party who owns input bit b_ω for wire ω has to secretly share it with the other participants. The input, in combination with the already secretly shared permutation bit λ_ω and signal strings $k_{\omega,0}$ and $k_{\omega,1}$, are the required information to choose the correct signal $k_{\omega,b_\omega\oplus\lambda_\omega}$ as garbled input. [4]

2.2.2. Phase 2. At first, all wire labels, gate labels, garbled input signals, as well as the permutation bits of the circuit output wires are sent to all participants [4]. After this, they can independently evaluate the circuit and obtain the calculation result at the output wires.

When a participant knows left input $k_{\alpha,a}$ and right input $k_{\beta,b}$ for a gate g, they can calculate $x_{\alpha,a}$ and $x_{\beta,b}$ by using G and compare the result with the public wire labels. The values a and b associated with the labels are known and thereby the evaluator now knows those same values a and b of the signals it holds. To obtain the gate output for

output wire γ, the party has to solve the previous equations for calculating the gate labels for the output signal

$$k_{\gamma,c} = \begin{cases} \bigoplus_{i=1}^{n}(y^i_{\alpha,0} \oplus y^i_{\beta,0}) \oplus X_g^{a,b}, \text{ if } a=0, b=0 \\ \bigoplus_{i=1}^{n}(z^i_{\alpha,0} \oplus y^i_{\beta,1}) \oplus X_g^{a,b}, \text{ if } a=0, b=1 \\ \bigoplus_{i=1}^{n}(y^i_{\alpha,1} \oplus z^i_{\beta,0}) \oplus X_g^{a,b}, \text{ if } a=1, b=0 \\ \bigoplus_{i=1}^{n}(z^i_{\alpha,1} \oplus z^i_{\beta,1}) \oplus X_g^{a,b}, \text{ if } a=1, b=1 \end{cases}$$

by using G again with the input signals. [4]

When an evaluator arrives at a circuit output wire ω, they can decrypt its signal $k_{\omega,\tau}$ by using the public permutation bit λ_ω to calculate $\tau \oplus \lambda_\omega$, in order to reverse (2). [4]

2.3. Protocol Improvements

- Wire labels can be omitted, as has been implemented by [5], because party i can simply compare $k^i_{\omega,a}$ with the shares $k^i_{\omega,0}$ and $k^i_{\omega,1}$ it owns and thus decide which of the signals it holds.
- Recent implementations of the protocol show that there exist more efficient alternatives for the earlier mentioned expensive zero-knowledge-proofs, without sacrificing any security [7].

2.4. Security of the BMR protocol

The BMR protocol is secure as long as honest parties, i.e., those who supply correct values for the computation, are in the majority [4].

Security against honest-but-curious adversaries, i.e., those who supply valid values but try to obtain private information, is guaranteed as long as at least one participant remains uncorrupted [1].

Since BMR is a cryptographic protocol, the security concept relies upon the assumption that adversaries can only act in polynomial time [4].

3. Applying the FreeXOR Optimization Technique to the BMR Protocol

In 2009, V. Kolesnikov and T. Schneider introduced the FreeXOR optimization technique for the two-party YGC protocol [8] and subsequent adaptation of it for the BMR protocol happened in [5].

FreeXOR essentially trivializes the construction and evaluation of XOR gates and thus can dramatically improve the runtime of both phases of the protocol. To accomplish this, garbled signal and gate layouts have to be modified. In this section, we are explaining how [5] did this.

The BMR FreeXOR adaptation from [5] only uses XOR and AND gates for its circuits. Here, AND gates are the only gates that still require gate labels and since XOR gates are negligible, circuits using as many XOR instead of other gates as possible are preferable.

Like many adaptations of the original BMR protocol do, that from [5] omits the usage of wire labels (see Section 2.3).

Note that in the following description of the modifications to the garbled circuit design, the usage of secretly shared values, like permutation bits, implies that during BMR protocol execution, the actual equations get securely evaluated on the shared values using SMC to retain secrecy.

3.1. Signal Modifications

The key idea that enables FreeXOR is understanding that making the values of the signal pairs of a wire dependent on each other does not invalidate security [8].

The signal share pair of party i for gate wire ω, *that is not an output wire of a XOR gate*, is created as

$$k^i_{\omega,1} = k^i_{\omega,0} \oplus R^i, \tag{6}$$

where R^i is of length κ. Here, $k^i_{\omega,0}$ remains random. R^i is party i's substring of the global value $R = R^i \cdots R^n$, called the *difference string*, which is created and secretly shared between parties the same way as the signals. The computation process of permutation bit λ_ω remains unchanged (see Section 2.2.1). [5]

The creation of *output wire signals for XOR gates* requires special care. Core of the optimization technique is that the computation of the output signal of a XOR gate does not require any gate labels [8].

Let an XOR gate g have input wires α and β, as well as output wire γ. Assume that wire α carries signal $k_{\alpha,u\oplus\lambda_\alpha}$ and wire β carries signal $k_{\beta,v\oplus\lambda_\beta}$, where u and v are the hidden semantics of the signals. The public output signal semantics for gate g is defined as the XOR of the input semantics. To enable this, permutation bit λ_γ is not random anymore, instead it is simply set as $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$ [5]. This permits that during circuit evaluation in phase 2, public output semantics can be obtained by calculating

$$(u \oplus \lambda_\alpha) \oplus (v \oplus \lambda_\beta) = (u \oplus v) \oplus (\lambda_\alpha \oplus \lambda_\beta) \tag{7}$$
$$= (u \oplus v) \oplus \lambda_\gamma. \tag{8}$$

As has been described in Section 2.2.2, values $u \oplus \lambda_\alpha$ and $v \oplus \lambda_\beta$ are known to an evaluator of the garbled circuit, if they hold signals $k_{\alpha,u\oplus\lambda_\alpha}$ and $k_{\beta,v\oplus\lambda_\beta}$ for the input wires of a gate.

Additionally, instead of being random, the *output signal pair* gets computed as

$$k_{\gamma,0} = k_{\alpha,0} \oplus k_{\beta,0} \text{ and } k_{\gamma,1} = k_{\gamma,0} \oplus R \tag{9}$$

respectively, where R is the aforementioned difference string [5]. The purpose of this is described in Section 3.2.2.

3.2. Gate Modifications

Since output signal creation differs between gate types, the gate design has to be modified depending on the gate type as well.

3.2.1. AND Gates. The computation of gate labels for AND gate g with input wires α and β, as well as output wire γ, happens according to (11) [5]. It essentially adapts the familiar gate label definition from Section 2.2.1 to the new signal design. Note that output signal $k_{\gamma,0}$ gets chosen for all labels and let $a = u \oplus \lambda_\alpha$ and $b = v \oplus \lambda_\beta$ be the public semantics of the input signals.

$$m = R \cdot (((a \oplus \lambda_\alpha) \cdot (b \oplus \lambda_\beta)) \oplus \lambda_\gamma) \tag{10}$$
$$X_g^{a,b} = F_{k_{\alpha,a},k_{\beta,b}} \oplus k_{\gamma,0} \oplus m \tag{11}$$

doi: 10.2313/NET-NET-2022-07-1_03

Here, $F_{k_{\alpha,a},k_{\beta,b}}$ is the processed output of a pseudorandom function. Its definition from [5] does differ from that of the pseudorandom generator in Section 2.2.1, but the purpose remains the same.

The calculation of m in (10) is the result of the fact that

$$u \wedge v = (a \oplus \lambda_\alpha) \wedge (b \oplus \lambda_\beta) \tag{12}$$

is equal to the multiplication of the bit values [5]. Now, two cases depending on parameter λ_γ have to be considered [5].

1) Let $\lambda_\gamma = 0$. If the result of (12) is 0, (10) yields $m = 0$ as well and the gate label masks $k_{\gamma,0}$. Otherwise, if the result of (12) is 1, (10) yields $m = R$ and the gate label masks $k_{\gamma,1} = k_{\gamma,0} \oplus R$. These are the expected results for an AND operation.
2) Let $\lambda_\gamma = 1$. If the result of (12) is 0, (10) yields $m = R$ and the gate label masks $k_{\gamma,1} = k_{\gamma,0} \oplus R$. Since $k_{\gamma,1}$ hides signal 0, this is as expected. Otherwise, if the result of (12) is 1, (10) yields $m = 0$ and the gate label masks $k_{\gamma,0}$. Again, the actual meaning of the signal $k_{\gamma,0}$ is inverted, i.e., 1 in this case, and the result is correct.

This shows that the AND gate correctly assigns the input signals to the corresponding output signals. Gate evaluation works by the same principle as we described in Section 2.2.2.

3.2.2. XOR Gates. XOR gates in the modified garbled circuits do not hold any labels. Let an XOR gate have input wires α and β, as well as output wire γ. An evaluator of the circuit simply has to calculate

$$k_{\gamma,(u\oplus\lambda_\alpha)\oplus(v\oplus\lambda_\beta)} = k_{\alpha,u\oplus\lambda_\alpha} \oplus k_{\beta,v\oplus\lambda_\beta} \tag{13}$$

on their input signals to obtain the output [5]. That this yields the correct output semantics for an XOR operation has been shown in (8). The following equations proof that the correct signals are calculated for all input cases [8].

$$\begin{aligned} k_{\gamma,0} &= k_{\alpha,0} \oplus k_{\beta,0} = (k_{\alpha,0} \oplus R) \oplus (k_{\beta,0} \oplus R) \\ &= k_{\alpha,1} \oplus k_{\beta,1} \\ k_{\gamma,1} &= k_{\gamma,0} \oplus R = k_{\alpha,0} \oplus (k_{\beta,0} \oplus R) = k_{\alpha,0} \oplus k_{\beta,1} \\ &= k_{\alpha,0} \oplus (k_{\beta,0} \oplus R) = (k_{\alpha,0} \oplus R) \oplus k_{\beta,0} \\ &= k_{\alpha,1} \oplus k_{\beta,0} \end{aligned}$$

They follow directly from the signal definitions.

4. Considerations for Using the BMR Protocol

In comparison to protocols like GMW and BGW, BMR's advantage of needing only a constant number of communication rounds for the circuit creation makes it a better fit for scenarios where communication between the parties is of comparatively more concern than local computation capabilities. For example, this is the case when SMC over the internet is required [5].

In scenarios with low network latency, protocols requiring a non-constant amount of communication rounds but less expensive processing, like the GMW protocol, could provide better overall performance than the BMR protocol [5].

It has to be noted that the original unmodified design of the BMR protocol is not suitable for real-world applications, because some details of it, like the usage of many zero-knowledge-proofs, are not efficiently computable, as has been noted by [7]. Since its introduction, however, much effort has successfully been spent to overcome those performance pitfalls. Eventually, reasonably efficient concrete real-world implementations of the protocol, like FairplayMP, introduced by A. Ben-David et al. in 2008 [6], have been developed.

5. Conclusion

We presented an expressive description of the basic BMR protocol for SMC, which enables the participation of any number of parties in a malicious setting. Its characteristic of requiring only a fixed constant number of rounds to create the garbled circuit makes it interesting for concrete real-world adaptations that use high latency communication over the internet. These implementations often introduce various optimizations to enhance its performance. The FreeXOR technique, for example, makes creation and evaluation expenses of XOR gates negligible in order to substantially boost performance.

References

[1] D. Evans, K. Vladimir, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018. [Online]. Available: http://dx.doi.org/10.1561/3300000019

[2] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, 1982, pp. 160–164. [Online]. Available: https://doi.org/10.1109/SFCS.1982.38

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[4] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 503–513. [Online]. Available: https://doi.org/10.1145/100216.100287

[5] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the internet," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 578–590. [Online]. Available: https://doi.org/10.1145/2976749.2978347

[6] A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: A system for secure multi-party computation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 257–266. [Online]. Available: https://doi.org/10.1145/1455770.1455804

[7] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant-round multi-party computation combining bmr and spdz," *Journal of Cryptology*, vol. 32, no. 3, pp. 1026–1069, Jul 2019. [Online]. Available: https://doi.org/10.1007/s00145-019-09322-2

[8] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 486–498. [Online]. Available: https://doi.org/10.1007/978-3-540-70583-3_40

Position-based Routing in Flying Ad Hoc Networks

Jakob Weigand, Florian Wiedner*, Jonas Andre*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: jakob.weigand@tum.de, wiedner@net.in.tum.de, andre@net.in.tum.de

***Abstract*—In recent years the interest in Flying Ad Hoc Networks (FANETs) to solve military and civil tasks increased significantly. Due to FANETs being comprised of highly mobile Unmanned Aerial Vehicles, the development of generally efficient routing algorithms proves to be considerably complex. This paper compiles a survey on routing protocols using position information in the routing process. Position-based routing protocols follow two main concepts, reactive and greedy-based. When comparing two algorithms, each following one of these concepts, different strengths and weaknesses become apparent, resulting in different ideal areas of application.**

***Index Terms*—FANET, position-based, routing, protocol, UAV, ad hoc network, MUDOR, GPMOR**

1. Introduction

With increased interest in using cooperating groups of Unmanned Aerial Vehicles (UAVs) to solve civil and military tasks, the concept of Flying Ad Hoc Networks (FANETs) was introduced to connect the individual UAVs as a mesh network. The existing FANET routing protocols follow two main strategies using either topology or position information in the routing process. This paper introduces position-based routing protocols for FANETs. It starts with outlining the reasons for the considerable difficulties in designing adequate routing algorithms, the specific characteristics, in Section 2. Subsequently, Section 3 outlines the two main approaches of position-based routing protocols, namely reactive and greedy-based. Thereafter, both Section 5 and Section 6 present one algorithm following each of these two approaches. This paper concludes with a comparison and assessment of the two presented algorithms in Section 7.

2. Characteristics of FANETs

FANETs consist of multiple highly mobile UAVs. This results in a specific set of characteristics, outlined and explained here:

Network topology: As the nodes of FANETs consist of individual UAVs, they possess a high degree of freedom in both speed and direction of movement. This results in a significantly reduced longevity of the network topology, especially when compared to ground-based networks [1], [2].

Node density: As UAVs do not require supporting infrastructure and are less likely disturbed by obstacles due to being located in the air, the relative node density can be assumed to be sparse [2].

Radio propagation model: Due to the high distances between nodes compared to the strength of the radio transmitters, FANETs usually require a free Line-of-Sight (LoS) between nodes. As FANETs are located in the air, they have a high likelihood of fulfilling this requirement [1].

Power sparsity: The availability of power to the routing protocol is highly dependent on the size of the used UAVs. For large UAVs the power required for routing calculations is insignificant compared to the power required for movement. For smaller UAVs the power capacity can be limited [1], [2].

3. Reactive and Greedy Routing Protocols

Position-based routing protocols in FANETs are generally separated into two distinct groups. Both of them are outlined here:

Reactive: In routing protocols using a reactive approach, the path discovery process is started on demand for every packet by flooding the network with a Route Request (RREQ) for a routing target. This is answered by a Route Reply (RREP) when the target was found. In comparison to proactively managing a routing table, this comes with a significantly reduced network overhead but usually increases the end-to-end delay. Compared to purely topology-based reactive routing protocols, the additional position information can be used to flood the network in a more controlled approach, reducing overall network overhead [1], [3].

Greedy: Greedy position-based routing protocols forward packets in the target direction without previously calculating a complete path to the target node either in a proactive or reactive manner [2], [3].

4. Related Work

Development of FANET routing protocols is a highly active field of research and as such a variety of related work is available. This paper provides an introduction to the topic of position-based routing algorithms by describing and comparing two algorithms, following fundamentally different approaches, in-depth. By contrast, Oubbati et al. [4] compile a more general, higher-level survey of position-based FANET routing protocols. Oubbati et al. [2], Lakew et al. [3], Sang et al. [5], and Perez et al. [6] compile generally broad surveys on routing

doi: 10.2313/NET-NET-2022-07-1_04

algorithms in FANETs. All of them reflect on different types of routing algorithms while dedicating chapters of their work to position-based routing algorithms. Oubbati et al. [2] should be highlighted in this context due to being extraordinarily extensive even compared to the other papers providing a general overview.

5. MUltipath DOppler Routing (MUDOR)

MUltipath DOppler Routing (MUDOR) is a reactive position-based routing protocol proposed by Sakhaee et al. [7] for FANETs. The main feature separating MUDOR from other reactive routing protocols is the incorporation of the relative node mobility to increase link stability and reduce the flooding overhead. The relative mobility of nodes is measured through observing the doppler shift of the received signals. MUDOR is partly based on Dynamic Source Routing (DSR) [8]. An optional Quality of Service (QoS) extension for MUDOR, proposed by Sakhee et al. [9], offers improved control over the required route performance. This section is based on the original MUDOR algorithm as proposed by Sakhaee et al. [7].

5.1. Physical Background

The doppler effect describes a perceived shift in frequency between the sender of a wave compared to the observer. As the velocity of UAVs is small compared to the speed of light [2], according to Rosen et al. [10] the aforementioned frequency shift can be approximated by:

$$\frac{f_o}{f_s} = 1 + \frac{v}{c} \tag{1}$$

With f_s being the frequency at the sending node and f_o being the frequency observed by the observing node. As f_s is standardized across all nodes, c is known and f_o is observed, the relative velocity v between sender and receiver can be calculated by solving Equation (1) for v:

$$v = c \cdot (\frac{f_o}{f_s} - 1) \tag{2}$$

Nodes approaching each other have a negative relative velocity and show a statistically higher link stability compared to receding nodes as they are longer in each others vicinity. Therefore, smaller values are superior. MUDOR introduces the Doppler Value (DV) metric representing the cost of each link based on this relative velocity. The DV is just the relative velocity calculated according to Equation (2) and weighted by -1 for approaching nodes and $+2$ for receding nodes:

$$DV = \begin{cases} -v, & v < 0 \text{ (nodes approaching)} \\ +2v, & v > 0 \text{ (nodes receding)} \end{cases} \tag{3}$$

5.2. Different Roles of Node

The MUDOR routing protocol differentiates between two different roles of nodes: requesting and receiving nodes. Each of the following sections describes one of them, with the role of the receiving node being subdivided in receiving Route Requests (RREQs) and Route Replys (RREPs). Roles are not node exclusive, therefore one node can have multiple roles.

TABLE 1: Format of a MUDOR RREQ as described by Sakhee et al. [7].

Name	Description
Request Id	Request identifier
Target Id	Target packet identifier
Hop Count	Hop counts until request termination
Packet Doppler Value	Largest doppler value on route
Route Addresses	Cumulated node addresses on route

5.2.1. Requesting Node. The communication process starts by the source node flooding its neighborhood with RREQs. The format of a RREQ is outlined in Table 1. Similar to other reactive routing protocols, a MUDOR RREQ possesses a maximum hop count field containing the maximum future hop counts until request termination, a unique request identifier, and a route addresses field cumulating the addresses of the previously visited nodes. Differences to other routing protocols occur in the target id and Packet Doppler Value (PDV) fields. Contrary to other routing protocols, the target of a MUDOR RREQ is a specific data packet, containing arbitrary data, instead of a node. The target id field contains the respective identifier of the target packet. The PDV field contains the largest DV observed during a node hop on the current route.

5.2.2. Receiving Node (RREQ). Each node possesses a request table containing all previously forwarded RREQ ids and the Minimum Doppler Value (MDV) of the specific requests. If a node receives a request, it compares the RREQ PDV with the observed DV in the last hop. The larger of the two values is then written into the PDV field of the RREQ. The following process differs depending on whether the node offers the requested data packet and if it has already forwarded the received RREQ.

Node offers the requested data packet: The node sends a RREP back to the last sending node. The RREP contains the same fields as the RREQ except for the target id and hop count fields which are omitted in the RREP.

Node has not already forwarded the RREQ: The node creates an entry in its request table containing the request id and the current RREQ PDV. Then it apprehends its address to the RREQ route addresses, decrements the RREQ hop count and forwards the RREQ to all neighboring nodes.

Node has already forwarded the RREQ: The node compares the current RREQ PDV with the PDV of the specific RREQ in its routing table. If the RREQ PDV is larger than the PDV in the route request table, the node has already forwarded the same RREQ on a superior route and the RREQ is dropped. If the current RREQ PDV is lower than the PDV in the request table, the newly discovered route is superior and the request table entry is overwritten with the RREQ PDV. Then the node apprehends its address to the RREQ, decrements the RREQ hop count and forwards the RREQ to all neighboring nodes. This measure enables the system to discover multiple routes leading through the same node while simultaneously reducing the overhead significantly compared to having a hard boundary of RREQs with the same id being forwarded by each node.

5.2.3. Receiving Node (RREP). The RREQ PDV is updated, as described in the previous paragraph, by comparing the RREQ PDV with the observed DV in the last hop and writing the larger value in the RREQ PDV field. The following process differs depending on whether the receiving node is also the requesting node:

Node is not requesting node: The RREP is forwarded by backtracking the route addresses in the corresponding RREQ field.

Node is requesting node: The node waits a configurable amount of time collecting incoming RREPs and ordering them by their PDV. Then it selects the path with the smallest PDV for packet transmission. If a selected path fails, MUDOR selects the path with the next smaller PDV. This is the multipath approach of MUDOR offering built-in failure recovery.

6. Geographic Position Mobility Oriented Routing (GPMOR)

Geographic Position Mobility Oriented Routing (GPMOR) is a greedy position-based routing protocol for FANETs proposed by Lin et al. [11]. This section is a summary of GPMOR as described in [11].

As outlined in Section 2, FANETs are a highly dynamic environment. Due to the high degree of mobility, a connection of two nodes can be interrupted during a packet broadcast even if they were initially sufficiently close. As the sender might not be able to detect such a loss of connection, this might lead to a significant amount of packet loss. GPMOR introduces an algorithm using the stored position and movement information to predict the future movement of potential relay nodes. Then, a node with a low probability of the described packet loss scenario is selected as the next relay node.

6.1. Mathematical Background

To predict the future velocity V_n and direction of neighboring nodes d_n, Lin et al. [11] use the following Gauss-Markov mobility model:

$$\begin{aligned} V_n &= \alpha V_{n-1} + (1-\alpha)\bar{V} + \sqrt{(1-\alpha^2)}V_{x_{n-1}} \\ d_n &= \alpha d_{n-1} + (1-\alpha)\bar{d} + \sqrt{(1-\alpha^2)}d_{x_{n-1}} \end{aligned} \tag{4}$$

with $\bar{V}$ and $\bar{d}$ being historical averages of V and d respectively. $V_{x_{n-1}}$ and $d_{x_{n-1}}$ are random variables from a Gaussian distribution introducing noise into the prediction equations. The tuning parameter α can be adjusted depending on the movement model, with $\alpha = 1$ representing no change of movement in the given time period.

The predicted values are then used by Lin et al. [11] to calculate the new position of the specific node after a time period ΔT:

$$\begin{aligned} x^{'} &= x + s_x \Delta T \\ y^{'} &= x + s_y \Delta T \end{aligned} \tag{5}$$

with x and y being the node coordinates and s_x and s_y being the velocity components in the respective dimensions calculated from V_n and d_n.

Lin et al. [11] then use the predicted node position to calculate the future euclidean distance between a relay node r and the destination node d:

$$\Delta d^{'} = \sqrt{(x_r^{'} - x_d^{'})^2 + (y_r^{'} - y_d^{'})^2} \tag{6}$$

If $\Delta d^{'}$ is below the range threshold R of node r, r will be able to send messages to the destination d after the ΔT used for the prediction calculation.

To decide which node the data packet is forwarded to if more than one node fulfills the $\Delta d^{'} \leq R$ prerequisite, the Metric To Connect (MTC) value is calculated:

$$\begin{aligned} \Delta x &= x_r^{'} - x_d^{'} & a &= (\Delta s_x^2 + \Delta s_y^2) \cdot R^2 \\ \Delta y &= y_r^{'} - y_d^{'} & b &= (\Delta s_x \Delta y - \Delta s_y \Delta x)^2 \\ \Delta s_y &= s_{ry} - s_{dy} & c &= \Delta s_x \Delta x + \Delta s_y \Delta y \\ \Delta s_x &= s_{rx} - s_{dx} \end{aligned}$$

$$MTC = \begin{cases} \frac{c-\sqrt{a-b}}{a}, & 0 < \Delta d \leq R \\ \frac{\sqrt{a-b}-c}{a}, & R < \Delta d \end{cases} \tag{7}$$

Source: [11]

The MTC value indicates the mobility relationship between nodes. In the first case $0 < \Delta d \leq R$, the nodes r and d are in range both before and after the prediction. This implies a strong correlation in movement and the assigned value is negative to display this condition. In the second case $R < \Delta d$, r and d are only in range in the predicted time step but not before. This signals a possible next hop but shows a historically worse movement corelation as compared to the first case, the assigned value is positive. In both cases the assigned absolute value is relative to the distance between nodes.

The outlined equations are only defined in a two-dimensional scenario, limiting the application area of GPMOR significantly.

6.2. Algorithm

The node discovery process works proactively by each node regularly sending HELLO messages to nearby nodes. These messages contain position and velocity information and are used by each node to maintain a node table. Any sent data packets contain the identifier of the destination node. Each intermittent node transmits the data packet to the best node according to the information in its node table. The algorithm terminates when the destination node is reached. The next hop is selected as follows:

1) The current source node calculates the immediate position of destination and neighboring nodes according to Equations (4) and (5).
2) The distance between destination node and all neighboring nodes is calculated according to Equation (6).
3) Now there are three distinct possibilities depending on how many neighboring nodes fulfill the $\Delta d^{'} \leq R$ condition:
   No node: The neighbors of the current source node are not directly in range of the destination node. An additional relay node is necessary. The neighbor with the smallest $\Delta d^{'}$ is selected.
   One node: This node will be in range of the destination node after ΔT. It is selected as next hop.

Multiple nodes: The MTC value between the affected neighboring nodes and the target node is calculated according to Equation (7). The node with the lowest MTC value is selected as next hop.

4) The next hop calculation algorithm is terminated and the packet is forwarded to the selected node. The selected node is the source node in the next iteration.

GPMOR only considers the currently best next relay node without considering the global situation with potential local but not global optima. This approach is considered greedy.

7. Comparison and Discussion

As MUDOR and GPMOR follow fundamentally different architectures, they also show significantly different characteristics. Table 2 shows an overview of these differences.

TABLE 2: Characteristics of MUDOR and GPMOR.

	TD	NO	BR	SPAR	SCAL
MUDOR	-	+	+	+	+
GPMOR	+	-	-	-	-

+: superior -: inferior

Transmitting delay (TD): Due to its greedy approach GPMOR has no notable transmitting delay as transmission starts instantly. By contrast, MUDOR has to calculate at least one complete route to the target node before being able to start transmission, leading to a significant transmission delay.

Network overhead (NO): As GPMOR has a reactive-based routing approach, exchanging the available network nodes regularly, it has a notably larger network overhead as compared to MUDOR.

Bandwith requirements (BR): As a consequence of its larger network overhead, GPMOR requires significantly more bandwidth as compared to MUDOR.

Network sparsity (SPAR): Due to its greedy approach a GPMOR RREQ can be stuck in a local but not global optimum. Currently GPMOR does not have any failure recovery strategy for this problem and therefore needs a sufficiently dense and convex network to avoid the occurrence of this problem. By contrast, MUDOR is not affected by sparse networks.

Scalabilty (SCAL): Due to its fundamentally proactive approach GPMOR has to manage a node table containing all available nodes in the network. This leads to a significantly worse scaling, especially concerning memory requirements compared to MUDOR which does not have to manage a similar node table.

The outlined significantly different characteristics also lead to different ideal areas of application. To have sufficient doppler shift to be able to avoid significant measurement errors, MUDOR is ideal for networks of fast and linear moving nodes such as larger scale fixed wing UAVs. Xi et al. [12] show that MUDOR does also work with slower moving ground-based nodes but the performance compared to other routing protocols increases with node speed. GPMOR needs a sufficiently dense network that does not violate its size boundaries. Compared to MUDOR it is superior in end to end delay. Therefore, it is optimally suited for dense networks of smaller scale UAVs such as grid focused search and rescue operations which also profit from its excellent end to end delay enabling in-person operation if necessary.

8. Conclusion and Future Work

The unique characteristics of FANETs pose a significant challenge for routing algorithms. This paper introduces a promising approach to solve this challenge, position-based routing protocols. These protocols can be categorized into two main strategies, reactive and greedy-based. A presentation and subsequent comparison of two algorithms following these strategies, MUDOR and GPMOR, shows distinct strengths and weaknesses. This results in complementary ideal areas of application with MUDOR showing superior characteristics for networks of fast moving fixed wing UAVs and GPMOR for dense networks of small scale UAVs.

Future work on the presented GPMOR algorithm could include a more sophisticated movement prediction model allowing the prediction of non-linear and three-dimensional movement. Additionally, the introduction of a route request failure recovery strategy is necessary to overcome the problem of a route request being stuck in a local but not global optimum.

References

[1] A. Chriki, H. Touati, H. Snoussi, and F. Kamoun, "Fanet: Communication, mobility models and security issues," *Computer Networks*, vol. 163, p. 106877, 2019.

[2] O. S. Oubbati, M. Atiquzzaman, P. Lorenz, M. H. Tareque, and M. S. Hossain, "Routing in Flying Ad Hoc Networks: Survey, Constraints, and Future Challenge Perspectives," *IEEE Access*, vol. 7, pp. 81 057–81 105, 2019.

[3] D. Lakew, U. Sa'ad, N.-N. Dao, W. Na, and S. Cho, "Routing in Flying Ad Hoc Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–1, 03 2020.

[4] O. Oubbati, A. Lakas, F. Zhou, M. Günes, and M. Yagoubi, "A Survey on Position-based Routing Protocols for Flying Ad hoc Networks (FANETs)," *Vehicular Communications*, vol. 10, 12 2017.

[5] Q. Sang, H. Wu, L. Xing, and P. Xie, "Review and Comparison of Emerging Routing Protocols in Flying Ad Hoc Networks," *Symmetry*, vol. 12, no. 6, p. 971, Jun 2020.

[6] A. Guillen-Perez, A.-M. Montoya, J.-C. Sanchez-Aarnoutse, and M.-D. Cano, "A Comparative Performance Evaluation of Routing Protocols for Flying Ad-Hoc Networks in Real Conditions," *Applied Sciences*, vol. 11, no. 10, p. 4363, May 2021.

[7] E. Sakhaee, A. Jamalipour, and N. Kato, "Aeronautical Ad Hoc Networks," in *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, vol. 1, 2006, pp. 246–251.

[8] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Comput.*, vol. 353, 05 1999.

[9] E. Sakhaee, A. Jamalipour, and N. Kato, "Multipath Doppler Routing with QoS Support in Pseudo-linear Highly Mobile Ad Hoc Networks," in *2006 IEEE International Conference on Communications*, vol. 8, 2006, pp. 3566–3571.

[10] J. Rosen and L. Gothard, *Encyclopedia of Physical Science*, ser. Facts on File Science Library. Facts On File, 2010.

[11] L. Lin, Q. Sun, J. Li, and F. Yang, "A Novel Geographic Position Mobility Oriented Routing Strategy for UAVs," *Journal of Computational Information Systems*, vol. 8, pp. 709–716, 02 2012.

[12] S. Xi and X.-M. Li, "Study of the Feasibility of VANET and its Routing Protocols," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1–4.

Survey on Trusted Execution Environments

Nicolas Buchner, Holger Kinkelin, Filip Rezabek*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: nicolas.buchner@tum.de, kinkelin@net.in.tum.de, frezabek@net.in.tum.de

***Abstract*—Confidentiality of code and data is an essential part of modern computing. As cloud services become more important as an easy way to use computational power, the need for keeping the exact code of the running applications from the companies that offer these services. A Trusted Execution Environment (TEE) is an option to remove the need for trusting the device the code is executed on and hide data from other processes. In this paper, the concept of a TEE is shown as well, as two implementations of TEE are being analyzed. First, a general overview of TEEs is given. Next, the functionality of Intel SGX and ARM TrustZone is being explained. Afterwards, the features of both implementations are shown, and the problems they have are being analyzed. Next, there is a comparison between the two implementations. Finally, other options to achieve trusted execution are being shown.**

***Index Terms*—trusted execution, privacy, security**

1. Motivation

In today's world, the need for security of the data used every day has become a significant factor in determining how to transport and process this data. Whilst looking at security during communication between different devices is common, the need to process data and execute proprietary code on client devices brought a new problem. The need to keep program code away from others has been the main reason these problems have become of more interest to companies. This is because the creation of programs has become more expensive and time-intensive, especially when Machine Learning models are a part of the application. This leads to the following questions. Does one trust the client's device? How does one execute code on data that the client's device should not have access to? Furthermore, is there a way to keep the client device's OS from accessing or influencing the execution of the application's code?

In recent years, research on ways to provide secure execution of code on untrusted devices has progressed. There are multiple ways to archieve the desired goal of code and data confidentiality on remote devices. Trusted Execution Environments (TEE) are one such option to provide the secure execution of code without interference from any other processes on the device. A TEE restricts access to the code and data of an application inside of it. Furthermore, it allows for verification of the TEE's content. A TEE requires some additional hardware on the device as well as software to manage the interactions. There are different implementations of TEE, and each comes with different strengths and weaknesses, some of which will be discussed later in this paper.

In this paper, the main components of a Trusted Executions Environment (TEE) are shown. Furthermore, two implementations of a TEE are being analyzed in terms of their functionality. These implementations are Intel SGX and ARM Trust Zone. Next, in Section 3 the features of both Intel SGX and ARM TrustZone are being shown. Afterwards, the problems of the respective implementation are analyzed. Finally, Section 4 shows different frameworks to help create applications intended for use with TEE.

2. Trusted Execution Environments

The TEE is a Secure Operating System separated from the original device's Operating System (OS). Additionally, it is supported by hardware components to provide the functions required by the applications, which are executed inside the TEE.

The features a TEE provides, depend on the individual implementations. Most commonly, a TEE provides some isolation of the processes from any process running in the normal OS of the device and from other processes inside the TEE. Furthermore, they allow verification of the executed code and data.

As shown by Arfaoui et al. [1], there are different ways to implement the hardware components of a TEE. The first option for hardware components of a TEE includes a trusted ROM, RAM, and a trusted processing environment. Furthermore, the TEE has its own crypto accelerators and can support trusted peripherals. The second option for the hardware components is to share the hardware with the regular OS and have a state that specifies if the currently executed process is trusted or untrusted.

The software part of a TEE is the TEE kernel, which is an OS, that is different from the host OS [1]. It is authenticated and validated during the start of the device and takes control upon the execution of the TEE application. Another software part is the TEE APIs. These APIs can be differentiated into private and public APIs. The private APIs provide a way for the Trusted Applications to use the functions provided by the TEE. The public APIs offer an interaction between applications running in the devices OS and the TEE applications.

Two such implementations are being shown in the following, and their functionality is being explained. Furthermore, the requirements of each implementation are being looked at.

doi: 10.2313/NET-NET-2022-07-1_05

2.1. Intel SGX

The first implementation of TEE looked at is created by Intel, called Software Guard Extensions (SGX). According to Intel [2], SGX provides isolation of program code and data in memory through hardware-based encryption of the memory. They claim this prevents more privileged processes, like the OS, from accessing this information. To use Intel SGX, the device needs to have an Intel CPU that supports SGX and have SGX enabled in the BIOS. The Intel CPUs that support SGX are all 6th to 10th generation processors as well as the server processors of the 11th and 12th generation. SGX is depricated in non server versions of th 11th and 12th generations due to lack of use cases for private users.

The hardware requirements of SGX are similar to the second option shown in Section 2. As such, the hardware used for SGX is primarily the original hardware, with some minor changes required. A Memory Encryption Engine is needed to input and output data from the TEE safely. The Memory Encryption Engine also stores the keys used for encryption of each secure memory area. Furthermore, SGX requires some additional microcode.

In SGX, the encrypted memory section and the respective key that belongs to it are called enclaves. Any user process that wants to use the SGX functions can create an enclave, as shown by Gu et a. [3]. On startup, the enclave is verified, usually through a hash, either on the local machine or remotely. The hash is 256 bits long and includes the code and the initial data of the enclave, as well as security flags and page locations within the enclave. This hash is then signed by either Intel or the program developer. If the developer signed the hash himself, the public key that is needed for verification of the signature has to be signed by Intel and added by the executing device to SGX [4]. The untrusted process, which the enclave belongs to, and the enclave share the same memory address space. The difference being that the enclave's memory is encrypted, as described in Jauernig et al. [5]. Any OS functionality, like memory management, interrupt handling, and I/O is still done by the device's OS. Still, neither the OS nor any other process can access or change any data or code inside the enclave's memory. When an enclave function is called, the encrypted memory section that belongs to this function is loaded into the CPU and then decrypted on the CPU for execution.

Any developer who wants to use SGX for their application has to create two parts. As shown in Figure 1, there is an untrusted part of the application and a trusted part, which is the enclave. The developer decides which functions of the application and which data require the enclave's security and then creates the enclave part out of it. The enclave needs to be verifiable, so a hash of the enclave's contents needs to be made. This hash is used to verify the enclave after the creation on the client device, most likely remotely by a server belonging to the program's creator. The untrusted part initializes the enclave on the client device, which is then verified. Once the application requires the contents of the enclave, the untrusted part calls the respective function of the enclave. The enclave then takes over and executes the called function and returns the output of this function. Then, the program's normal execution continues until an enclave function is required again. For any communication with the enclave that contains data, a secure channel is created by the enclave, and the enclave includes its verification for the other process to verify the data is coming from or going to the correct enclave [4].

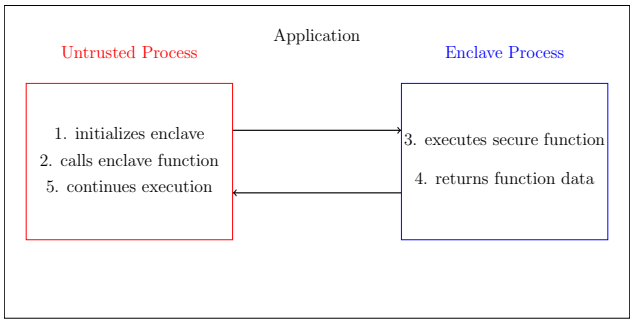


Figure 1: Application using SGX enclave

2.2. ARM TrustZone

The other implementation being analyzed is called TrustZone. It is created by ARM [6]. Similar to SGX, TrustZone's hardware components are the ones described as the second option in Section 2. In TrustZone, the execution environment is split into two parts: the secure world and normal world. These two worlds are not just different terms used by ARM to describe the same thing as untrusted and enclave parts in SGX. While the untrusted part is about the same as the normal world, an enclave does not represent the same as the secure world in TrustZone. To use TrustZone, an ARM processor with TrustZone support is required. The ARM processors that the Cortex-A and Cortex-M classes of their processors.

The hardware changes required for TrustZone are very minimal. The memory controller needs to be able to differentiate between secure and normal world processes. The device's OS is considered a normal world process in this context. The CPU needs to be able to do the same differentiation between secure and normal world. This ability to differentiate keeps anything running in the normal world from interfering with secure world applications. The context switch between these two worlds is done by trusted firmware on the device [5].

On the software side, there is a separate OS that runs in the secure world, unlike in SGX. During the device's boot, the image of TrustZone is verified and authenticated. This is not done for individual application initializations as all the applications run in the same TEE instance for TrustZone, while in SGX, each enclave is a separate TEE instance. Thus any program in the secure world can influence other programs in it. In order to deal with this problem, an application can only be added to the TrustZone of a device if the device's creator allows this application's inclusion.

A developer who wants to use TrustZone can decide to create a secure world-only application. Developers can also choose to develop both an application that runs in the normal world and have an application in the secure world for the security-relevant features.

doi: 10.2313/NET-NET-2022-07-1_05

3. Features and Problems

In the following, different implementations of TEE are being analyzed with a focus on Intel SGX and ARM TrustZone. The features of each implementation are being shown, and finally, the problems of the implementation are being analyzed.

3.1. Intel SGX Analysis

As described in Section 2.1, SGX enclaves are encrypted while they are in the device's memory. Thus an attacker can not read any of the contents of an enclave. Furthermore, the attacker can not access the encryption key, as it is stored on the Memory Encryption Engine. The attacker could still change any part of the encrypted enclave, but the enclave is verified when it is used, which means the attack would be noticed. The enclave can be initialized again to match the actual content the developer intended. These features result in the fact that the program's developer no longer needs to trust the entire device the program runs on. The developer's trust must now only be in the CPU that the code is executed on. SGX provides separation between processes executed inside of enclaves as well, which means the execution of one enclave can not impact any other enclave. However, one feature is not provided by SGX, which is support for trusted peripherals. The fact that this feature is missing means that no device directly connected to the client can influence anything in the enclave or be used directly for I/O purposes.

As for the disadvantages of using SGX, the enclave needs to be decrypted each time its functions are executed. After execution of the enclave function, the entire enclave needs to be encrypted again before it can be stored in the device's memory. Both of these operations increase the application's execution time on the client device. Depending on the size of the enclave and the frequency of calls to enclave functions, this en-/decryption time can be a significant part of the overall execution time of the application. Furthermore, Section 2.1 shows the need for the developer to split the application into two parts and also create means to verify the content, i. e. creating a hash of it and signing the enclave. This increases the development time of such an application.

To summarize, SGX provides an environment to execute code without the interference of other processes on the device and guarantees the confidentiality and integrity of both code and data, as described by Narra et al. [7]. Furthermore, it allows for the remote verification of the created enclave. As for the drawbacks of SGX, the development effort is more significant than making a regular program. On the client-side, the processing overhead increases because the enclave needs to be decrypted for execution and the verification of the data entering the enclave. Dinh Ngoc et al. [8] describe the preformance overhead based on the different calls in SGX and the amount of CPU cycles each of them takes. Finally, SGX does not protect against side-channel attacks. Thus some information about the program can still be inferred.

3.2. ARM TrustZone Analysis

In Section 2.2, TrustZone was shown to have the secure world and the normal world. The secure world is also described to be the part of TrustZone, which holds the TEE part. In the secure world, the applications are shielded from any influences coming from the normal world. Thus no normal world process can read or change the content of the memory areas belonging to any secure world application. Neither can any normal world process interfere in the execution of code belonging to the secure world. Should the attacker have access to the hardware itself though, TrustZone does not offer any protection. This comes from the fact that the memory is not encrypted like it is in SGX, and the context switch is done by the trusted firmware. In TrustZone's case, the developers' trust needs to be put into the TrustZone firmware and the additional hardware parts of TrustZone. However, in contrast to SGX, TrustZone does offer support for trusted peripherals by including the drivers for the peripherals in the secure OS [5]. Unlike SGX, there is no en-/decryption needed for execution, only a context switch that takes very little time to do.

The disadvantages of using TrustZone are, as previously mentioned the fact that there is no separation between applications running in the secure world. Additionally, TrustZone only protects against software attackers and does not protect against hardware-level attackers, as there is no encryption of the secure memory parts. Furthermore, the developers of applications, which are running inside the TrustZone of a device, need to trust each other because of the missing separation. Finally, TrustZone is only verified on boot of the device and does not verify applications before they are executed.

In summary, TrustZone offers an execution environment for code without the influence of processes in the normal world. The integrity of the secure world is checked on boot, and the contents of the secure world are kept confidential from the normal world but not from the secure world. In terms of drawbacks, TrustZone only protects against software attackers. Furthermore, there is no separation between programs inside the secure world, forcing developers to additionally trust other applications in the device'sTrustZone. Similar to SGX, TrustZone does not protect against side-channel attacks.

TABLE 1: SGX and TrustZone features

	Intel SGX	**ARM TrustZone**
Trusted Part	CPU	Firmware and Hardware
Protection against Hardware Attacks	yes	no
Separation of TEE applications	yes	no
Verification of Trusted application	remotely or locally when called	only on system boot (entire TEE not individual applications)
Trusted Peripherals	no	yes

4. Applications

After understanding how Intel SGX and ARM TrustZone work and what they offer, we take a look at how to create applications that use either SGX or TrustZone.

doi: 10.2313/NET-NET-2022-07-1_05

For Intel SGX, there are multiple frameworks, which are meant to help develop an application for it. There are two kinds of frameworks for SGX. The first of them is intended for the creation of new applications which use SGX. The other type of framework can make an existing application run in an SGX environment.

Intel themselves created one framework that is of the first kind, and it is called SGX SDK [9]. SGX SDK is a framework that supports C and C++ programming languages, and its development tools can be used on both Windows and Linux operating systems. It does not only include the libraries for SGX but also contains tools for debugging and some examples to help understand how to use this framework. Intel actively updates the SGX SDK.

Another framework for creating new TEE applications is Open Enclave SDK [10]. Open Enclave is an open-source framework for TEE applications. Same as SGX SDK, it works with C and C++ languages and has versions for both Windows and Linux. Unlike SGX SDK, though, it does not only work for developing applications intended for Intel SGX. It also allows for creating programs designed for use on ARM TrustZone. Open Enclave is also regularly updated by multiple authors.

The other kind of framework is used to make an already existing application run in an SGX enclave. There are multiple commercial frameworks for this purpose, like Fortanix [11]. Furthermore, there are some open-source frameworks like Graphene [12]. They work based on LibOS. To use Graphene, the host it should run on needs to support SGX SDK. The application is signed together with the Graphene enclave part to form the enclave. Then the created enclave is sent to the host it should run on together with Graphene.

Both of the shown TEE implementations are also available for use on different devices. While ARM TrustZone is mainly used on mobile devices, Intel SGX is aimed for use on client pcs and servers. Some cloud service providers already include support for SGX on their platform. One such Cloud service is Microsoft Azure Confidential Computing [13]. Microsoft Azure supports applications using SGX SDK, Open Enclave and some other frameworks used to create SGX programs. Furthermore, it allows for remote attestation of the enclaves. On Azure, there is also support for another kind of TEE called AMD SEV, which was not presented in this paper.

5. Related work

There are other papers, which are looking into different implementations of TEE. One such paper is Jauernig et al. [5], describing five different TEE implementations there. The implementations described are Intel SGX, AMD SEV, ARM TrustZone, as well as the two academic TEE implementations, Sanctum and Sanctuary. They are described in terms of their functionality and the provided features.

Other concepts can provide security features to a device. One such concept is the Trusted Platform Module (TPM). A TPM is a hardware module that contains some data storage capacity, as well as the hardware required to generate both symmetric and asymmetric encryption keys and can create cryptographic hashes. Aaraj et al. [14] explain that a TPM offers cryptographic functions as well as protected storage to perform integrity checks on the platform or any application that is using it. The TPM itself is only encrypted storage for keys in a hierarchical system of keys, some of which are bound to data on the system's storage, and others are just used to keep the bound keys safe. The TPM is only a cryptographic co-processor and cannot be used for general computation.

Another option for trusted execution is smart cards. Smart cards are small chips protected from the physical environment and usually break if one tries to overcome the physical protection of the chip. A smart card has a small connection interface, and some smart cards can even be accessed remotely. A typical example of a smart card is a credit card. Naccache and M'Raihi [15] explains, that the connection interface of a smart card is standardized and smart cards do not possess a power source of their own. In essence, a smart card is a small computer that is powered as long as it is connected to another device and can be authenticated against the connected device or remote users and then do some computation on the smart card. As a smart card is equipped with cryptographic functions, all of the data entering and leaving the card can be secured as well. There are two types of smart cards, cryptographic smart cards, that only offer cryptographic functions like authentication or encryption. The other type of smart card is called the java smart card. This second type of smart card allows for more general computational use of the computer inside.

6. Conclusion

In conclusion, we looked at what a TEE is and what is required to implement a TEE and then analyzed two different implementations. The requirements for a TEE are split into hardware and software parts, and there exist multiple ways to implement a TEE in both of these parts. Of the implementations looked at, each has its respective advantages and disadvantages. SGX is built only to require minimal hardware changes and relies primarily on encrypting memory areas. These areas are decrypted only for the CPU upon execution of the contained code and can not be read or altered without detection. The enclaves in SGX are also kept separate to prevent one from influencing another.

In TrustZone, on the other hand, the separation in different worlds is mainly achieved through the trusted firmware performing a context switch. In Section 2.2, we saw that TrustZone allows for secure peripherals, in contrast to SGX, but it does not separate applications running in the secure world.

Finally, other ways to achieve trusted execution on untrusted devices are shown. Each of these options has its own individual features and problems, each with slightly different use cases. TPM offers cryptographic operations on a coprocessor; smart cards can either be used for cryptographic functions only or be a general-purpose execution environment on a small card that needs to be connected to another device for power.

A TEE does not guarantee that an attacker can not gain information about the code or data executed inside the TEE despite the features provided. The vulnerabilities of specific implementations could be an exciting topic for further research, as well as general problems of TEE.

doi: 10.2313/NET-NET-2022-07-1_05

Another subject for further research could be an analysis of the performance overhead of different implementations. Different implementations use different mechanisms to achieve their features and have very different execution times.

References

[1] G. Arfaoui, S. Gharout, and J. Traoré, "Trusted execution environments: A look under the hood," in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2014, pp. 259–266.

[2] [Online]. Available: https://www.intel.de/content/www/de/de/architecture-and-technology/software-guard-extensions.html

[3] Z. Gu, H. Jamjoom, D. Su, H. Huang, J. Zhang, T. Ma, D. Pendarakis, and I. Molloy, "Reaching data confidentiality and model accountability on the caltrain," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 336–348.

[4] [Online]. Available: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation

[5] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: Properties, applications, and challenges," *IEEE Security Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

[6] [Online]. Available: https://developer.arm.com/ip-products/security-ip/trustzone

[7] K. G. Narra, Z. Lin, Y. Wang, K. Balasubramaniam, and M. Annavaram, "Privacy-preserving inference in machine learning services using trusted execution environments," *arXiv preprint arXiv:1912.03485*, 2019.

[8] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about intel sgx performance on virtualized systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–21, 2019.

[9] [Online]. Available: https://01.org/intel-softwareguard-extensions

[10] [Online]. Available: https://github.com/openenclave/openenclave/tree/master/docs/GettingStartedDocs

[11] [Online]. Available: https://fortanix.com

[12] [Online]. Available: https://graphene.readthedocs.io/en/latest/oldwiki/Introduction-to-Graphene-SGX.html

[13] [Online]. Available: https://docs.microsoft.com/de-de/azure/confidential-computing/enclave-development-oss

[14] N. Aaraj, A. Raghunathan, and N. K. Jha, "Analysis and design of a hardware/software trusted platform module for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 8, no. 1, pp. 1–31, 2009.

[15] D. Naccache and D. M'Raihi, "Cryptographic smart cards," *IEEE micro*, vol. 16, no. 3, pp. 14–24, 1996.

Review of Industrial Control Systems Protocols

Alexandru Cruceru, Lars Wüstrich* and Patrick Sattler*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge54dov@mytum.de, wuestrich@net.in.tum.de, sattler@net.in.tum.de

***Abstract*—This paper provides an overview of the actual state of the art of industrial control systems protocols. ICS protocols are data transfer protocols used for the communication between devices working in an industrial control system. The protocols are classified based on two criteria: whether a protocol is vendor-specific or not and regarding the industrial sector in which the protocol is used. The paper also presents in more detail the characteristics and the design features of some popular ICS protocols.**

***Index Terms*—industrial control system (ICS), ICS protocol, process automation, building automation, power-grid automation, meter-reading automation**

1. Introduction

Industrial Control Systems (ICS) are nowadays a highly important component of large-scale producing companies and factories, from manufacturing lines and building automation to power grids, water treatment facilities, and transportation systems. Critical infrastructure, on which the comfort and wellbeing of entire cities or regions rely, is dependent on such systems which must operate with high precision and performance for keeping up to the requirements needed in such industrial fields.

ICS are in use for over forty years but have evolved and changed due to the growing requirements. The basic tasks of ICS are to gather information from remote sensors, to evaluate the collected data, to give commands to the singular components (e.g. valve, pump, turbine, burner, industrial machine) of the system, and to provide a Human-Machine Interface. As these systems grew larger and larger and as the requirements became more complex, remote access to ICS through the internet became a must. Thus, there were developed industrial systems, which work with wired and wireless connectivity using Ethernet, Routing, and IP.

The interconnectivity and communication between ICS devices are represented as an industrial network that has, in general, other performance goals than usual network systems used for Internet communication. Reliability and real-time operations are critical in such industrial networks, low bandwidth and latency have to be aimed so that data availability is a high standard. [1] Therefore, serial connection and specialized protocols focusing on specific functionality stood at the core of ICS for a long period, Ethernet being later introduced as a need to integrate ICS to the Internet. Migration towards Ethernet and IP exposed ICS design vulnerabilities. The focus on time performance pushed aside data integrity and confidentiality.

The present article discusses the ICS protocols, offering classification, and exemplification in detail. The structure of the present article is: Chapter two presents an overview of the existing ICS protocols; Chapter three focuses on a detailed description of four ICS protocols; and the last part presents the conclusion of the article.

2. Overview of the existing ICS Protocols

ICS protocols have a long history. They have been deployed starting with the first ICS devices more than over forty years ago. ICS protocols were designed to work with serial communication and with high real-time performance. Since ICS devices are used in various industrial fields, a variety of specialized ICS protocols have been developed. While certain protocols are specialized only for an industrial sector, other protocols can be implemented in more than one field.

This paper focuses on two ICS protocol classification criteria: (1) vendor-specific or widely used ICS protocols; (2) industrial sector based.

TABLE 1: ICS Protocol Classification

Protocol	Vendor-specific	Sector
Modbus	No	PA
HART-IP	No	PA
Profinet/Profibus	No	PA
FOUNDATION Fieldbus	No	PA
EtherCAT	No	PA
EtherNet/IP	No	PA
CIP	No	PA
Siemens S7	Yes	PA
Sinec H1	Yes	PA
FINS Omron	Yes	PA
DNP3	No	PA/PGA
ICCP	No	PGA
BACnet	No	BA
Niagara Tridium Fox	Yes	BA
ANSI C12.22	No	MRA
OSGP	No	MRA/PGA

PA=Process Automation
PGA=Power Grid Automaton
BA=Building Automation
MRA=Meter Reading Automation

2.1. Vendor-specific or widely used ICS protocols

Vendor-specific ICS protocols are designed by companies that are also ICS device manufacturers. They are designed to work only with devices produced by the same

doi: 10.2313/NET-NET-2022-07-1_06

company or to integrate devices from multiple manufacturers. Big players in the automation device market are Tridium, Omron, Siemens, Schneider Electric, and Rockwell Automation, all being also ICS protocol developers. Protocols like Niagara Tridium Fox were developed for Tridium devices, OMRON FINS for Omron devices, Siemens S7 and Sinec H1 for Siemens devices, Foxboro for Schneider Electric devices, and CISP for Rockwell Automation. [2]

Widely used ICS protocols are non-proprietary, so they can be used on devices from different manufacturers and comply with the performance standards demanded in most of the ICS. Some of these are Modbus, BACNet, HART-IP, EtherNet/IP, EtherCAT, ProfiNet/Profibus, DNP3, and ICCP.

2.2. Industrial sector based

ICS are used in various industrial sectors having specific requirements. Therefore, each ICS protocol was developed to implement use-case-specific features and to perform data transfer operations for a distinct industrial field. This paper concentrates on a classification done previously [3], to divide the ICS protocols into specific industrial fields.

2.2.1. Process Automation. Process automation is the broadest field in which ICS are used. It concerns the use of automation devices in factories and firms so that production and administration processes are controlled and monitored using a computer infrastructure. Multiple industries use process automation, like the automotive industry, chemical industry, oil refining industry, gas industry, water industry, and wastewater industry. The main benefits of process automation are to reduce personal costs and to increase productivity. Process automation consists of the integration of multiple input and output devices in a centralized system. This system is then controlled using a computer infrastructure that comes through graphical user interfaces in contact with human administrators. The input devices are sensors that measure different production parameters (e.g., temperature, pressure, volume) and the output devices are controlled units like valves, pumps, or motors that perform different tasks. These devices are connected to programmable logic controllers (PLC) which receive the data from the sensors, processes it, and then send commands to the controlled units. The PLCs are connected with each other and also with control computers that monitor and supervise the entire production process. The control computers are accessed by human operators that can coordinate the processes from here. ICS protocols are responsible for the data transfer between all these devices. There are many protocols developed for supporting this type of ICS. Widely used protocols (Modbus, FOUNDATION Fieldbus, Profibus/Profinet, CIP - with its implementations ControlNet, DeviceNet and EtherNet/IP), EtherCAT and HART-IP) and vendor-specific protocols (FINS Omron and the Siemens protocols- Siemens S7 and Sinec H1) were developed for providing data transfer between devices that work in a Process Automation ICS. DNP3 was originally developed for power grid automation but is nowadays also used for process automation.

2.2.2. Building automation. Building automation describes the automated, centralized control of the HVAC (heating, ventilation, and air conditioning), lighting, access control, and fire detection systems of a building. Building automation systems are used in both commercial buildings as well in private homes. They consist of multiple sensors and output devices, which work together with the computer infrastructure. The sensors gather information from the environment, send it to controllers which analyze the data received and give commands to the output devices. (In general, humans can also intervene through an HMI.) An example is the fire extinction system. Smoke and temperature sensors send the data to the controllers which analyze it and determine that a fire has broken out in a specific room. The controllers then stop the elevators, isolate the area where the fire is burning by closing the doors and start the watering system. Important to note is that, in general, all these devices are built by different manufacturers and use different software. Therefore, ICS protocols deal with the integration of building automation devices into one system, providing them with a standardized data transfer format. BACnet and Niagara Tridium Fox are protocols used for Building automation.

2.2.3. Power Grid Automation. Power grid automation is used to supervise and automatically control the power system using ICS devices. The protocols specialized in this sector deal with communication between different power stations and communication within one station. An automated power system has three tasks: data acquisition (the system acquires data through measuring devices and stores it), supervision (administrators and engineers analyze the data together with the computers and check if everything works as expected), and control (the computers or the operators of the system send instructions to power-system devices). One station can also receive or send the acquired data to another remote station so that outages are better monitored. The power system automation supervises the whole process, from the generation of electrical power to the delivery of it to the consumers. [4]

Protocols specialized in this industrial sector are DNP3, ICCP, and IEC 61850 and IEC 60870-5 standards.

2.2.4. Meter Reading Automation. Meter reading automation consists of automated transfer and centralized storage of data from metering devices that measure utility consumption of households, businesses, and institutions. Automated utility meters measure the use of resources and store it. They then use an ICS protocol to send the data through a network within regular time intervals to the data center of the utility provider. This data is then analyzed to check if the meter works fine and to calculate the consumption of the customer. In many situations, the customers are also provided with access to the data through the internet. The automation of meter reading has multiple benefits for both providers and consumers. The providers can reduce their personnel costs and monitor the devices remotely, and the customers can manage their consumption by having access to the consumption information. The most important protocol used for this purpose is ANSI C12.22. OSGP (Open Smart Grid Protocol) is another protocol that operates in both Meter Reading and

Power Grid automation and it is used in the deployment of electrical smart meters. [5]

3. Characteristics and Design of the most used ICS Protocols

This section includes a detailed presentation of the design and features of the frequently used ICS protocols: Modbus, ICCP, BACnet, and DNP3. The first three protocols are each representative of a different industrial sector. DNP3 is used in both process automation and power grid automation.

3.1. Modicon Communication Layer

Modicon Communication Layer (Modbus) [6] is an application layer protocol designed by Modicon (later bought by Schneider Electrics), first deployed in 1979. Modbus operates by using a master-slave architecture. There are two cases: either a Human Machine Interface (HMI) that acts as a master and multiple Programmable Logic Controllers (PLC) acting as slaves, or a PLC acting as a master having other devices, like sensors, motors, or other PLCs as slaves. Master devices can be, at the same time, slaves of other devices. The master/slave architecture is based on a request/reply methodology, where a master sends a request to the slaves and the slaves send a reply to that request. Masters can send either broadcast messages that address all slaves or individual messages that address an individual slave. The slaves cannot send a message unless they received a request that was addressed to them. [1] Modbus uses 3 distinct Protocol Data Units (PDU) for communication: Modbus Request, Modbus Response, and Modbus Exception Response. The master sends a Modbus Request at the slave including a Request PDU. The slave receives the request and responds to it either with a Data Response in the PDU if there is no error occurred, or with a Modbus Exception Response if an error occurred during the transmission. [1]

Being an Application Layer protocol, Modbus can be easily adapted to either serial or routable network protocols. RS-232 and RS-485 are used on the physical layer for serial communication. Ethernet is used on the physical layer for networked communication while IP and TCP are used as protocols for the Link and Transport Layer. In time, different variants of Modbus were developed, three of which are Modbus RTU, Modbus ASCII, and Modbus TCP. Modbus RTU and ASCII are used in asynchronous serial communication while Modbus TCP is used for routable communication. Modbus TCP has two solutions for integrating a Modbus message to the routed Internet. It either adds a Modbus Application Protocol header, which includes Link and Transport layer information to the existing serial frame keeping the original address information and error check, or it removes the original address information and error check, keeping the Modbus PDU and attaching the Modbus Application Protocol header to it. The first solution is commonly implemented in legacy devices. The second one is preferred in the implementation of modern devices. [1]

Since Modbus was designed for serial communication and time performance, it lacks some features that are important when using the Internet. Modbus has no authentication procedure and uses no encryption. It also allows, in some cases, the serial networks to be flooded with messages due to no broadcast suppression. [1]

3.2. Inter-Control Center Communication Protocol

The Inter-Control Communication Protocol (ICCP) [7] was developed by a working group founded in 1991, tasked by the International Electrotechnical Commission to create a standardized real-time data exchange protocol, which should facilitate the communication between electric power utility stations. ICCP is an application layer protocol. It was designed to support a set of data transfer operations between electric control centers. These operations are: establishing a connection with other control centers, reading and sending information from and to remote centers, configuring and controlling remote devices, and controlling programs on remote centers. [1]

The ICCP is based on a client-server architecture. The server center contains data and functions which are accessed by the client center via a request. Most of the implementations of the ICCP allow nowadays that a device is both a server and a client.

The transfer procedure of the ICCP uses a bilateral table which takes the role of an access control list. The bilateral table has the purpose of checking the access rights of the client that requests access to data or control. Therefore, it strictly defines what information is accessible to which control center. To ensure that the access rights are agreed upon by both centers, the bilateral table entry must match on both server and client. [1]

ICCP is a wide-area network protocol. Since it operates at the application layer, it can work with different transport and link-layer protocols and use different physical media. ISO transport on port 102/TCP over Ethernet is mostly used for the implementations of this protocol. [1]

Like other ICS protocols, ICCP also lacks authentication and encryption, leaving this in the hand of lower layer protocols. ICCP is highly accessible as it operates on wide-area networks, therefore it is susceptible to denial of service attacks. [1]

3.3. Building Automation and Control Networks

BACnet [8] stands for Building Automation and Control Networks, and it was first presented by the American Society for Heating, Refrigerating, and Air-Conditioning Engineers in 1987. Buildings have nowadays a lot of facilities offered by HVAC, access control, lighting control, elevator, and fire alarm devices. All these devices are produced by multiple manufacturers and thus use different operating programs and protocols. BACnet was developed for integrating all these devices into a single control system so that building owners do not have to be dependent on one manufacturer or do not have to use a different management system for every device.

BACnet uses an object-oriented model for data transfer between system devices. The Information shared between the devices is represented as a logical object. These objects are abstract constructs that are characterized by a set of

properties. They describe physical inputs, outputs, or non-physical components like software. An example for such an object would be a logical representation of a temperature measuring device which has as property the value of the measured temperature. The use of objects organizes the information and standardizes the data formats that can be transmitted. BACnet defines a set of 25 standardized object types that offer usage in a wide area of applications. It also allows the vendors to customize these objects by adding specific properties to them or to create entirely new objects. [9]

Besides the objects used for data representation, BACnet also provides standardized services. They are responsible for the interaction within the system, describing actions that can be performed by a device. BACnet provides a wide functionality through these services, grouped into the following categories: object access, alarm and event management, scheduling, trending, file configuration and transfer, and device management. [9]

BACnet offers support for a variety of network implementations. The most used ones are BACnet/Ip (which uses Ethernet and IP) and a low-cost implementation called MS/TP (Master-Slave Token Protocol) (which uses RS-485 together with twisted pair cable). MS/TP networks are used to couple devices that transmit a low volume of data, and which do not require high transfer speeds. BACnet/Ip is used for high-speed transmission of larger data blocks, providing also interface for data that has to be routed outside the current Local Area Network segment using IP. Specialized BACnet controllers and routers are responsible for organizing and controlling the infrastructure of a BACnet network.

3.4. Distributed Network Protocol

The Distributed Network Protocol (DNP3) [10] was developed by Westronic in 1990. It was designed for communication within the electric power industry, in environments with high electromagnetic interference but implemented in other industries as well. [1]

DNP3 is based on a master-slave architecture similar to Modbus. In contrast to Modbus, it allows bidirectional communication: master–slave, slave-master. In addition, DNP3 puts a high accent on reliability. To ensure reliability DNP3 uses many cyclic redundancy checks (CRC), one for the link-layer header and one for every 16 payload data bytes. If errors are identified by the receiver when checking the CRC, the message is retransmitted. Besides this, DNP3 provides an acknowledgment mechanism to prevent the loss of frames due to physical layer errors. The sender of the message requests the receiver to send a confirmation that the message was received. If no such confirmation is received by the sender, it sends the message again. These two safety mechanisms provide high reliability but also a higher overhead, a fact that represents a problem in some real-time environments. [1]

DNP3 supports multiple data types: files, counters, analog, and binary data, and other types of data objects. The data is structured into multiple data classes. Class 0 stands for static data. This data type is used for representing the current values that the supervised objects gathered, providing the master with a real-time view of the monitored system. Classes 1 to 3 stand for event data. Event data is time-stamped and prioritized, class 1 having the highest priority and class 3 the lowest. Event data represents old data stored in the buffer of a remote terminal unit (RTU). Through the time-stamp, event data offers a historical view of the system. Unsolicited reporting is another feature that DNP3 has. In contrast to Modbus, DNP3 allows slave-stations to send messages without getting a request from their masters. This feature allows slaves to send messages immediately as an event occurs and they do not have to wait for the master to request their data. Unsolicited reporting makes the system more efficient but adds overhead to the message frames. As slaves initiate communication and the master acknowledges receiving the message, the frames have to include both source and destination address. [1]

DNP3 was integrated into Internet communication by adding an IP and TCP or UDP header to the DNP3 frame. For a more secure Internet connection, Secure DNP3 was developed. This version of the protocol provides an authentication mechanism. Authentication is initiated by the receiving device. When the sender tries to access data from the receiver, the receiver requests identification of the sender before giving him access to the data. [1]

4. Conclusion

ICS and ICS protocols represent a major topic in the domain of distributed systems. Automation is an important tool in the present-day industry. Due to high product demand, productivity and efficiency become a must. Infrastructure is continuously growing and becomes more complex in order to satisfy people's needs. Without automation, the productivity of such complex systems would be low, control and monitoring nearly impossible. ICS protocols are essential components of automation, being responsible for the transfer of information between industrial devices.

ICS protocols are designed for various industries, as such a variety of such protocols exists. They can be either developed by ICS device manufacturers (to serve the devices these manufacturers build) or can be designed for the general use of any ICS devices specialized in a certain industry. The four industrial fields that use ICS protocols are Process Automation, Building Automation, Power Grid Automation, and Meter Reading Automation. Because every industry has different performance standards, protocols must be adapted to the special needs of every industry. Therefore, ICS protocols are, in general, specialized for an industrial sector, with few exceptions.

As seen in the third chapter, the design of ICS protocols differs from one protocol to another. Client-Server architecture, Master-Slave architecture, or an object-oriented methodology are three examples of design choices. In general, these protocols operate on the application layer of the ISO/OSI model and therefore are compatible with many implementations on lower layers. Ethernet and TCP/IP were added to the ICS protocols as a need to integrate the ICS devices into the Internet.

References

[1] E. Knapp and J. Langill, *Industrial Network Sekurity, 2nd Edition*. Syngress, 2014.

[2] Dragos, "ICS & IT PROTOCL SUPORT," https://www.dragos.com/wp-content/uploads/Dragos-Supported-Protocols.pdf, 2021.

[3] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "An internet-wide view of ics devices," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, 2016, pp. 96–103.

[4] Wikimedia Foundation, "Power-system automation — Wikipedia," https://en.wikipedia.org/wiki/Power-system_automation, 20 September 2021, online, accessed on 19 December 2021.

[5] ——, "Smart meter — Wikipedia," https://en.wikipedia.org/wiki/Smart_meter, 22 November 2021, online, accessed on 19 December 2021.

[6] Modbus Org., "The Modbus Organization," https://modbus.org/, online, accessed on 19 December 2021.

[7] "Telecontrol equipment and systems - Part 6-503: Telecontrol protocols compatible with ISO standards and ITU-T recommendations - TASE.2 Services and protocol," International Electrotechnical Commission, Standard, Jul. 2014.

[8] ASHRAE, "BACnet Website," http://www.bacnet.org/index.html, online, accessed on 19 December 2021.

[9] D. Fisher and PolarSoft, "How BACnet is Changing Building Automation Networking," *The Extension. A Technical Supplement to Control Network*, vol. 8, 2007.

[10] DNP Org., "DNP.org," https://www.dnp.org/, online, accessed on 19 December 2021.

doi: 10.2313/NET-NET-2022-07-1_06

Applications of Q-Learning to Network Optimization and Graph Problems

Marco Dollinger, Max Helm, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: dollingm@in.tum.de, helm@net.in.tum.de, jaeger@net.in.tum.de

***Abstract*—This paper provides a theoretical overview of Markov Decision Processes (MDP), Reinforcement Learning (RL) in general, and (Deep) Q-Learning in particular. Furthermore, we examine the application of Deep Q-Learning in network optimization of Software-defined Satellite-terrestrial networks and in general graph problems like the traveling salesman problem (TSP) and a graph representation of the boolean satisfiability problem (SAT). Furthermore, we reference the results obtained by Deep Q-Learning approaches to the examined application areas. Moreover, we give an overview of recent research progress in the field of Reinforcement Learning and present open questions and challenges.**

***Index Terms*—Q-Learning, Reinforcement Learning, Software-defined Satellite-terrestrial Networks, SAT, TSP**

1. Introduction

In the recent past, Reinforcement Learning has received extensive research interest from academia as well as industry. In particular, Reinforcement Learning promises meaningful advances in the field of robotics, control theory, statistics, and economics among others. The Google DeepMind application AlphaGo, which is based on Reinforcement Learning, was even covered by mainstream media for beating world-class players in the board game Go. This paper is structured as follows: Section 2 provides an introduction to the theoretical foundations of Reinforcement Learning, and in particular to Q-Learning which is a specific type of Reinforcement Learning. Section 3 analyzes and categorizes applications of Q-Learning to graph problems like the traveling salesman problem and the boolean satisfiability problem as well as applications to network optimization on the example of Software-defined Satellite-terrestrial networks. In Section 4, an overview of recent developments and future challenges of Reinforcement Learning research is given. Lastly, Section 5 concludes the paper and summarizes the results.

2. Theoretical Foundations

In this section, an overview of the theoretical foundations of Reinforcement Learning and specifically Q-Learning is presented.

2.1. Reinforcement Learning

Reinforcement Learning is one of the three main paradigms of modern machine learning, next to supervised and unsupervised learning. In Reinforcement Learning, an agent takes actions within an environment to maximize rewards. Usually, a Reinforcement Learning environment is modeled as a Markov Decision Process. In [1], Watkins defined an MDP as:

- a set of actions `A`
- a set of states S,
- $R_a(s, s')$: a reward function that rewards the agent when transitioning from state s to state s' using action a
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$: a probability function that expresses the probability that a certain action a which is taken at time t in state s will result in state s'.

It is important to note that Markov Decision Processes satisfy the Markov Property that transitions and rewards only depend on the current state and are independent of previous actions or states.
Further, we will only introduce finite Markov Decision Processes with finite sets of actions and states. The model of an MDP is the combination of transition function and reward function. When the model of an MDP is unknown, Reinforcement Learning is a possible technique to find an optimal policy ("when to choose which action"). As described by Kaelbling et al. in [2], we can divide RL techniques into model-free approaches and model-based approaches.

- Model-free: learn a policy without learning the model.
- Model-based: learn the model to derive a policy.

The goal for the RL agent is to find a policy that maximizes the acquired rewards until a sequence of actions leads to a final state, or the algorithm is aborted (e.g. by a time constraint). Figure 1 shows the general framework for Reinforcement Learning of a Markov Decision Process as illustrated by Wang et al. in [3].
In a Reinforcement Learning process, the term "regret" describes the performance differences between the actual agent and a (hypothetical) optimal agent which we aim to minimize. A common obstacle to minimizing regret is the tradeoff between long-term rewards (exploration) and short-term rewards (exploitation). Exploitation means learning from previous experiences and choosing the most optimal action as the next decision. In contrast, exploration is the process of trying new policies/decisions that can offer a smaller immediate reward but enables the agent to learn more information about the environment [2].

doi: 10.2313/NET-NET-2022-07-1_07

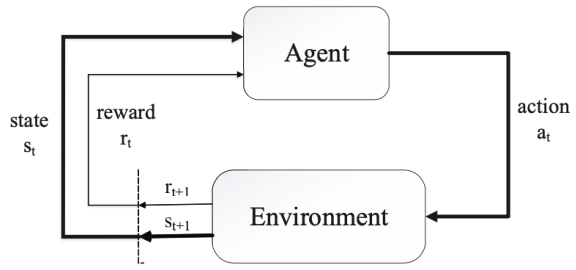


Figure 1: The Reinforcement Learning Framework [3]

2.2. Q-Learning

Q-Learning is a model-free Reinforcement Learning algorithm that learns a controller without learning transition probabilities or rewards from the environment. The Q-function calculates the quality of a state-action combination which is a measure of how good it is to take an action in a specific state.

$$Q : S \times A \to \mathbb{R} \tag{1}$$

The learning goal is to find an optimal Q-function for each state-action pair that can be used to achieve our goal of maximizing reinforcement rewards. At the beginning of learning, the Q-function might be randomly initialized. During learning, the agent will update the Q-function with each decision step with the following formula [1]:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2}$$

where:

- r_t is the reward that our agent receives when moving from state s to state s_{t+1} using action a
- the learning rate $\alpha : 0 < \alpha \leq 1$ defines how much we weight "new knowledge" compared to previous experiences. This is similar to many machine learning algorithms.
- the discount factor $\gamma : 0 < \gamma \leq 1$ weighs immediate rewards against future rewards
- $\max_a Q(s_{t+1}, a)$ is an estimate of the maximum reward that can be obtained from state s_{t+1}

When implementing the Q-Learning algorithm in an RL agent, it would be understandable that for every action decision, the agent should choose the action with the highest Q-value for the current state. However, with this approach, the agent is purely exploiting previous knowledge and neglects the exploration aspect of the RL task. This problem is solved by ϵ-greedy Q-Learning as proposed by Wunder et al. in [4]. In ϵ-greedy Q-Learning, the agent chooses the action with maximum Q-value with the probability of $(1 - (\epsilon(k - 1)/k)$ and selects one of the remaining actions with a uniform probability distribution. With increasing ϵ, the agent increasingly explores the environment rather than exploiting its knowledge. However, ϵ needs to be sufficiently small to avoid unnecessarily increasing the learning duration.

Since in basic Q-Learning (2), we calculate the Q-function for the whole, (sparse) action-state-matrix, the algorithm can become very computationally expensive. To speed up our computations we can use Quantization which means grouping similar actions or states together at the cost of quantization errors. Quantization can discretize infinite spaces or decrease the cardinality of discrete state/action spaces. Another approach to handle large action and state spaces is function approximation. With function approximation, the agent does not compute the complete action/state matrix, but rather only "estimates" the Q-function values, for example by using neural networks. [3]

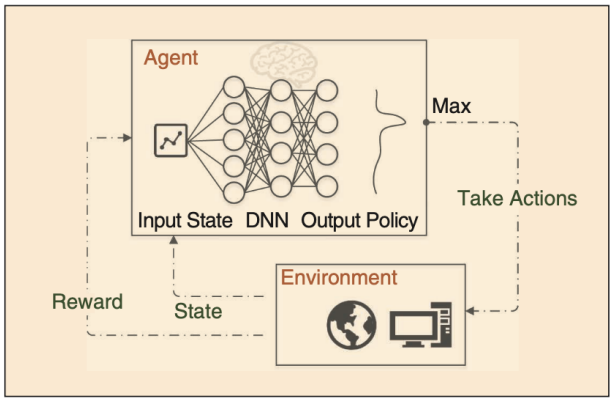


Figure 2: An illustration of DQL. DNN: deep neural network [5]

2.3. Deep Q-Learning

Deep Q-Learning (DQL) uses a neural network to realize non-linear function approximation which enables efficient Q-Learning in high dimensional action and state spaces. Figure 2 shows the general Reinforcement Learning framework with a neural network agent as illustrated by Tham et al. in [5]. The input of the neural network represents the current environment state, whereas the maximum value of the output layer encodes the next action to take by the agent.

3. Applications of (Deep) Q-Learning

This section introduces example applications of (Deep) Q-Learning to network optimization and graph problems. Since the examined problems are very high-dimensional and computationally expensive, (Deep) Q-learning promises great performance in their respective solutions.

3.1. Network Optimization of Software-Defined Satellite-Terrestrial Networks (SDSTN)

One goal of network optimization is the optimal network resource allocation to many different actors. This matching problem is a high-dimensional task that can be modeled as an MDP and therefore solved with Reinforcement Learning. In other words, network optimization aims to fulfill user requirements/requests while minimizing resource consumption.
As Chao et al. showed in [6], Deep Q-Learning can

doi: 10.2313/NET-NET-2022-07-1_07

improve network resource allocation in software-defined satellite-terrestrial networks. The physical resources of the network are categorized in the data layer as networking capabilities through low earth orbiters (LEO), caching/storage space in distributed infrastructure and computing capacity of distributed mobile edge computing (MEC servers). Software-defined satellite-terrestrial networks virtualize physical resources by a logically centralized control layer that allocates the optimal resources to user/application requests like communication or navigation tasks [7]. Since the physical network resources are distributed over many different entities, allocating specific devices to a user request is a high-dimensional matching problem that can be modeled by the general reinforcement framework in Figure 1 and thus solved by Deep Q-Learning. The data layer of the network represents the state $S(t)$ at time t of the RL task, in particular the position and availability of LEOs, the cached contents, and the idle/occupied computing capabilities. The RL agent is the control layer that decides for a given user u request at time t the optimal action $a_u(t)$ which includes assigning an LEO, deciding if the requested content should be cached, and allocating a MEC server to execute the computation task. According to He et al. in [8], the control layer needs to pay fees to consume the physical resources, whereas the user u pays the control layer for executing a request. The RL reward function calculates the ratio of fees paid by the control layer divided by the fees paid by the user. With increasing efficiency of physical resource consumption, while maintaining constant user fees, the respective fee ratio increases which rewards the agent to optimize its allocation policy [6].
To measure the performance/efficiency of the Deep Q-Learning approach to resource allocation, Chao et al. simulated an SDSTN with three LEOs, five MEC servers, and five content caches. Compared to a static resource-to-user allocation strategy, the DQL-based resource allocation achieved greater utility per resource and thus increased the efficiency of the network. Additionally, the authors showed that "with the increase of training episodes, the expected utility per resource increases" [6] which proves that the modeled reward function can be used to optimize the agent's policy, and therefore optimize the network's allocation strategy.

3.2. Graph Problems

3.2.1. Boolean Satisfiability Problem (SAT). Given a boolean formula, the Boolean Satisfiability Problem is the task of finding a satisfying variable configuration. Since SAT is NP-complete, commercial solvers rely on heuristics to speed up finding a satisfying configuration or proving unsatisfiability. As shown by Kurin et al. in [9], Deep Q-Learning can potentially be applied to commercial settings and reduce wall-clock time to solve SAT problems. In [9], Kurin et al. introduce Graph-Q-SAT that uses Q-Learning with graph neural networks as function approximation. Similarly to Selsam et al. in [10], boolean formulas in conjunctive normal form (CNF) are represented by bipartite graphs. Since boolean formulas vary in size and the graph changes during solving by setting variables, the DQL input must support dynamic dimensionality. Therefore, Graph-Q-SAT uses Graph Neural

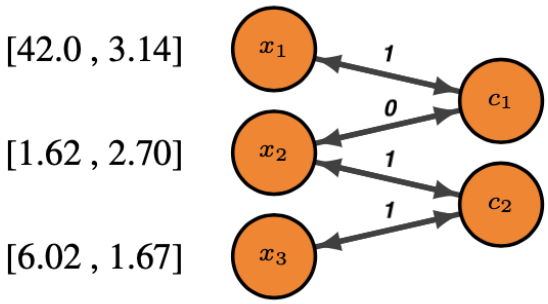


Figure 3: Graph representation of $(x_1 \vee x_2) \wedge (-x_2 \vee x_3)$. The annotations are Q-function values for setting variables to true or false respectively [9].

Networks as formalized by Battaglia et al. in [11]. Besides vertices and edges, Graph Neural Networks include annotations, which change by their operations. For example, Figure 3 shows a possible input/output of a Graph Neural Network of the formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. To decide the next action (setting one variable to True or False), the agent selects the highest annotation value of all variable nodes. In the example of Figure 3, the agent will set $x_1 = True$. The reward function punishes the agent for each non-terminating decision which incentivizes the agent to find a satisfying configuration as fast as possible. For unsatisfiable formulas, Graph-Q-SAT will try all configurations to prove unsatisfiability [9]. To evaluate Graph-Q-SAT, Kurin et al. trained the agent with Random 3-SAT instances from the SATLIB benchmark. It is demonstrated that Graph-Q-SAT outperforms Variable State Independent Decaying Sum (VSIDS) by reducing the required iterations to solve SAT problems by 2-3 times. VSIDS is a frequently used Conflict Driven Clause Learning (CDCL) branching heuristic which means that for each iteration the solver chooses a variable and assigns a binary value, similarly to Graph-Q-SAT. In particular, Graph-Q-SAT needed less than half the iterations of VSIDS for SAT-50-218 instances (50 variables, 218 clauses). Further important characteristics to evaluate are the generalization properties of Graph-Q-SAT. Compared to VSIDS, "Graph-Q-SAT has no difficulty generalizing to larger problems, showing almost 4X improvement in iterations for a dataset 5 times bigger than the training set" [9] which shows great generalization to other problem sizes of Graph-Q-SAT. Another important characteristic is the generalization to unSAT problems (unsatisfiable SAT instances) when trained only on SAT problems. While Graph-Q-SAT can solve unSAT problems, its performance is worse than on SAT problems relative to VSIDS. This is partly because unSAT differs from SAT problems, where proving unsatisfiability requires exhausting all possible assignments, whereas one satisfying assignment suffices to prove satisfiability. While Graph-Q-SAT achieved great performance overall, Kurin et al. noted that "more work is needed to apply Graph-Q-SAT to reduce wall clock time in modern SAT solving settings" [9].

3.2.2. Travelling Salesman Problem (TSP). Given a weighted graph where vertices represent cities, the traveling salesman problem is the task of finding the shortest

doi: 10.2313/NET-NET-2022-07-1_07

Hamiltonian circle. Since the TSP is NP-complete, we rely on heuristic algorithms to efficiently solve the problem for large graphs. Introduced by Gambardella et al. in [12], the Reinforcement Learning-based algorithm "Ant-Q" can be applied to the TSP and achieve competitive performance compared to other heuristic algorithms.
Ant-Q is inspired by the "ant system" (AS) as described by Colorni et al. in [13], and the Q-Learning algorithm [1]. Because Ant-Q is a distributed algorithm, the performance can be further increased by additional (distributed) computing capacity. Given an agent k located in city r, the agent moves to city s using the following formula:

$$s = \underset{u \in J_k(r)}{argmax}[AQ(r,u) \cdot HE(r,u)] \tag{3}$$

where:

- $J_k(r)$ is the list of cities that agent k has not yet visited.
- $AQ(r,u)$ is the equivalent to the Q-function (2) that expresses the usefulness of moving to city u when located in city r
- $HE(r,u)$ is a heuristical value that is used to prefer small distances. E.g. by multiplying the inverse of the distance between r and u

While Ant-Q was the best performing algorithm "compared to the elastic net, simulated annealing, the self-organizing map, and farthest insertion" [12] for specific standard sets of the symmetric TSP (symmetric weights), Ant-Q's polynomial time complexity makes it impossible to apply it to large TSPs. However, for asymmetric TSPs, which are harder than the symmetric case, Ant-Q delivered promising results that are usually only achieved by very specialized algorithms [12].

4. Recent Developments in Deep Reinforcement Learning Research

Hardware advances, particularly GPUs, have driven increased interest in Deep Learning over the last decade. As mentioned in Sections 2 and 3, Deep Neural Networks are used for function approximation in Deep Reinforcement Learning. Since function approximation made it feasible to apply Reinforcement Learning to increasingly large action and state spaces, DRL continues to be successfully applied to fields like robotics, control theory, and more. Recent examples were the super-human performance DRL agents achieved in playing Atari games [14] or the success of DeepMind's AlphaGo achieving world-class performance in the board game GO [15], which is computationally considerably more complex than chess. Another example is the OpenAI Gym Bipedal Robot, which successfully applies DRL to the control of robotic joint angles which has analog, or potentially very large, state and action spaces [15].
However, the increasing applications of DRL have also raised further problems and open questions which continue to drive research interest. For example, how to secure the stability of DRL, which refers to the stability of weights of the DNN, remains a mostly open question [16]. Another research area is the application of transfer learning to speed up the training process. When training a robotic DRL agent with visual input, transfer learning enables the agent to learn from simulated data before using real-world data which makes training more cost-efficient and speeds up development cycles [14].

5. Conclusion

This paper provides a theoretical introduction to Reinforcement Learning and in particular Q-Learning as well as the techniques to apply Q-Learning to high dimensional data through function approximation. Additionally, Section 3 summarized possible applications and results of Deep Q-Learning (function approximation with deep neural networks) to network optimization and graph problems. As shown in Section 3, Deep Q-Learning outperformed static resource allocation strategies in the simulation of Software-defined Satellite-terrestrial networks [6]. Additionally, Deep Q-Learning promises to reduce wall-clock time in SAT-solving [9] and asymmetric TSPs [12]. Section 4 outlined recent research activity in reinforcement learning and raised open questions/challenges to further apply Deep Q-Learning in fields like robotics.

References

[1] C. Watkins, "Learning From Delayed Rewards," 01 1989.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *CoRR*, vol. cs.AI/9605103, 1996. [Online]. Available: https://arxiv.org/abs/cs/9605103

[3] H. Wang, X. Chen, Q. Wu, Q. Yu, X. Hu, Z. Zheng, and A. Bouguettaya, "Integrating Reinforcement Learning with Multi-Agent Techniques for Adaptive Service Composition," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 12, pp. 1–42, 05 2017.

[4] M. Wunder, M. Littman, and M. Babes-Vroman, "Classes of Multiagent Q-learning Dynamics with ϵ-greedy Exploration," 08 2010, pp. 1167–1174.

[5] M.-L. Tham, A. Iqbal, and Y. Chang, "Deep Reinforcement Learning for Resource Allocation in 5G Communications," 11 2019, pp. 1852–1855.

[6] Q. Chao, H. Yao, F. Yu, F. Xu, and C. Zhao, "Deep Q-Learning Aided Networking, Caching, and Computing Resources Allocation in Software-Defined Satellite-Terrestrial Networks," *IEEE Transactions on Vehicular Technology*, 04 2019.

[7] B. Yang, Y. Wu, X. Chu, and G. Song, "Seamless Handover in Software-Defined Satellite Networking," *IEEE Communications Letters*, vol. 20, 06 2016.

[8] Y. He, F. Yu, N. Zhao, and H. Yin, "Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep Reinforcement Learning Approach," *IEEE Communications Magazine*, vol. 55, pp. 31–37, 12 2017.

[9] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning," *CoRR*, vol. abs/1909.11830, 2019. [Online]. Available: http://arxiv.org/abs/1909.11830

[10] D. Selsam, M. Lamm, B. Bunz, P. Liang, L. Moura, and D. Dill, "Learning a SAT Solver from Single-Bit Supervision," 02 2018.

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, and ..., "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[12] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem." 01 1995, pp. 252–260.

[13] A. Colorni, M. Dorigo, and V. Maniezzo, "An Investigation of some Properties of an "Ant Algorithm"." 01 1992, pp. 515–526.

[14] K. Arulkumaran, M. P. Deisenroth, and ..., "A Brief Survey of Deep Reinforcement Learning," *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[15] J. Shin, T. Badgwell, K.-H. Liu, and J. Lee, "Reinforcement Learning – Overview of Recent Progress and Implications for Process Control," *Computers & Chemical Engineering*, vol. 127, 05 2019.

[16] L. Busoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, 10 2018.

doi: 10.2313/NET-NET-2022-07-1_07

Seminar Innovative Internet Technologies: Zero Knowledge Proofs

Sebastian Hohl, Filip Rezabek*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: sebastian.hohl@in.tum.de, frezabek@net.in.tum.de

***Abstract*—In this paper the general idea and properties of zero knowledge proofs are discussed. The modern zero knowledge proofs zk-STARKs, zk-SNARKs (Ligero and Sonic) and Bulletproofs are compared regarding their need for a trusted setup, proof length, proving time and verification time. Zero knowledge proofs are (non-)interactive proofs that yield no information to the verifier. They are widely useable as they exist for every problem in NP. Zero knowledge proofs are used for many applications like signatures, anonymous decentralized payments on blockchains or verifying computations.**

***Index Terms*—zero knowledge proof**

1. Introduction

How much information have to be used to perform a certain action and how to minimize the released information? Any information given away might be used for attacks and other malicious activities, so information minimization is a fundamental security principle [1]. Especially if the aim is to convince someone of something, minimizing the released information seems to be hard. Zero knowledge proofs provide a solution for this problem. The idea of zero knowledge proofs (ZKPs) shown in [2] is that *they guarantee that a polynomial time verifier gains essentially no information from a proof.*
The verifier is only convinced that the given statement is valid. The prover does not release its secret information. So zero knowledge is a property a proof has. This is a way to show that a proof does not reveal too much information. [2, 3]
As ZKPs are well suited for the use on blockchains, they have got more attention with the recent rise of blockchain technology [4].
In this paper a part of the general theory about ZKPs is introduced in Section 2. Then modern ZKP systems are considered in Section 3 and in Section 4 applications of ZKPs are discussed. In the last Section 5 related works are recommended.

2. What are Zero Knowledge Proofs?

This section introduces to the basics of ZKPs. They have an extensive theory, so more advanced topics cannot be discussed in this brief paper. After reading this section the reader will know the meaning of every part of the term *(non-)interactive zero knowledge argument of knowledge*, therefore this section prepares the reader for the used terms of Section 3. At first in Subsection 2.1 the parts of a ZKP system are discussed. Then the essential properties of every ZKP are introduced in Subsection 2.2. Furthermore, differences between interactive and non-interactive ZKPs and how non-interactivity is achieved are considered in Subsection 2.3. After that the meaning of the suffix "of knowledge" is explained in Subsection 2.4. Finally, in Subsection 2.5 a possible use of the functionality of ZKPs is discussed from a blackbox perspective.

2.1. Framework

The statement to be proven is a decision problem. It consists of a language L[1] and for any given input x one has to decide whether $x \in L$. A ZKP system consists of such a decision problem and two programs[2]: One is the *verifier* and has typically limited resources like a polynomial runtime limit for its calculations. The other one is the *prover* that either has more computational power or usually a witness for the problem. The used language and the input x are known to both verifier and prover, so the secret information is the witness allowing to prove the membership of x efficiently. [2, 3, 6, 7]
Polynomial runtime limits in this Section 2 are taking the size of the common input $|x|$ as argument of the polynomial limiting the runtime.
For interactive ZKPs the verifier and prover communicate in alternating rounds or in the case of non-interactive ZKPs (NZKPs) the prover creates one proof that can be verified without further interaction (see Subsection 2.3). The purpose of the prover is to convince the verifier that $x \in L$ is correct without revealing more information. The task of the verifier is to check if $x \in L$ is correct with sufficiently high probability. Both prover and verifier can use randomness, so a recording of the view of the verifier of a proof is a random variable. [2, 3, 6, 7] This framework is illustrated in Figure 1.

2.2. Properties of a ZKP

A ZKP system fulfills the following three properties:
Completeness:

1. A language is a set of words over an alphabet.
2. Anything like algorithms or Turing machines or equally computationally powerful is considered according to the Church-Turing Hypothesis [5].

doi: 10.2313/NET-NET-2022-07-1_08

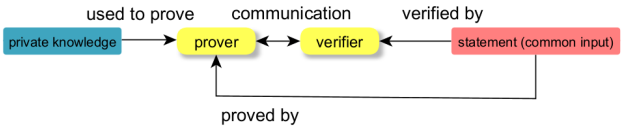


Figure 1: The prover has to convince the verifier that the statement is valid.

Completeness means that for $x \in L$ the prover having a witness or enough computational power can/will convince the verifier of this correct statements. This convincing can either be successful with sufficiently high probability or in case of *perfect completeness* with certainty. This assumes that the verifier fulfills its part of the protocol[3]. [2, 3, 6]

Soundness:

If the statement is not correct (if $x \notin L$) the verifier should only be convinced by *any* program[4] with a sufficiently low probability. This means that wrong statements are not likely accepted. Also note that this is not defined as chance of zero to convince the prover of a wrong statement. This property is often weakened to *computational soundness* like in Section 3. In this case the soundness property only has to hold against programs that have polynomial runtime. Therefore any program with more runtime could convince the verifier of at least one false statement with a higher probability. Often ZKPs with computational soundness are called zero knowledge *arguments* instead of ZKPs [3, section 2.1], like the ZKPs mentioned later in Section 3. But the term ZKPs in this work includes zero knowledge arguments aswell. [2, 3, 6]

Zero Knowledge:

For the zero knowledge property soundness and correctness are not regarded. The prover is the same as in the ZKP system, but the verifier can be any potentially adversarial program that might try to get additional information. The idea of [2] was that the *no information*[5] gain of any verifier can be defined by the notion of indistinguishability between the distribution[6] of the recorded view[7] of this verifier and another distribution that is made by a simulator running in expected polynomial time. This simulator only assumes that $x \in L$ and does not use the prover at all, so that the conclusion is that this verifier cannot gain anything new from the proof it could not compute in (expected) polynomial time on its own. [2, 3]

There are many variants and slightly changed definitions for this, but two common are:

- Perfect zero knowledge: Both distributions are identical and thus indistinguishable.
- Computational zero knowledge: Both distributions cannot be distinguished by any algorithm with polynomially bounded computation time except with a negligible small probability difference.

Note that perfect zero knowledge implies computational zero knowledge. Computational zero knowledge is the most general and therefore often used as a synonym for zero knowledge. [2, 3, 8, 9]

One disadvantage of this definition of *no information* gain is that following these definitions every interactive proof for a problem in polynomial time (P) is a ZKP, as the verifier/simulator could solve it in polynomial time on its own. So zero knowledge makes only sense for problems that do need more than (expected) polynomial time, therefore the name *zero knowledge proof* may be a bit misleading.

There are many variants [3] and slight variations of these definitions, here only the most basic can be mentioned.

Another more restrictive variant is *honest verifier zero knowledge* [3, Section 3], which means that there only has to exist such a simulator for the one verifier of the ZKP system (e.g. any dishonest/cheating verifier may get information).

A more restrictive definition is *auxiliary input zero knowledge*, where the simulator and the verifier both get access to a string of already known knowledge. This means that no previous knowledge can be used by a verifier to gain more information from the interaction than it could compute with this previous knowledge by itself. Auxiliary input zero knowledge implies that the composition of ZKPs stays a ZKP. [10]

A less restrictive variant is that of *witness indistinguishability*, where the used witness from a set of possible ones cannot be determined. This can be combined with witness hiding (e.g. no new witness can be computed from the proof). [11]

2.3. Non-Interactive ZKP

Interactivity is sometimes too difficult or too costly to achieve. This is especially important for blockchains (or other similar structures) as very relevant use case of non-interactive ZKPs (NZKPs), where the proofs must be publicly verifiable. This means that the proof must be a one-way message created by the prover, that is then saved on the blockchain. This proof must be verifiable by any participant of the blockchain without any interaction with the prover, only using the blockchain data. [6] [4].

Only very limited languages like those in BPP[8] have NZKPs [10]. Therefore the used model has to be changed. As in many ZKPs the verifier only sends randomly selected challenges to the verifier, this random selection could be done by a trustworthy public source of randomness instead. So the prover and verifier would both know the resulting random challenges, but there would be no need for the verifier to send these questions to the prover. The verifier would still receive all answers to these challenges in the one-way proof of the prover. [7, 12]

An initial approach is to have a shared random bit string per statement [13] that allows the creation of a ZKP that can be verified non-interactively. As common random strings do not simply exist, there are the following two approaches. For each approach common data is generated in an interactive setup at the start of the ZKP system. After this setup the system can be used for non-interactive

3. A "troll" verifier could always reject.
4. This means not only the prover of the system, but every program that can take its place.
5. except that the statement is true
6. It is a distribution as both programs might use randomness.
7. This includes everything it can read from and the communication.
8. In BPP are problems that can be solved with high probability in probabilistic polynomial time.

doi: 10.2313/NET-NET-2022-07-1_08

proofs, even by non-participants of the setup.
One option is the *Common Reference String* model, where from a distribution a common string can be generated [14]. But this requires a *trusted setup* of the distribution. The trusted setup of a NZKP system consists of several participants[9] that each generate their own secret and use these in an interaction that is used to create the distribution. In the best case these individual secrets are destroyed immediately. But if anyone gets all those secrets (for example if all participants collude) the soundness and the zero knowledge property are no longer guaranteed [7, 14]–[16].
Another option to get NZKPs is to use the *Fiat-Shamir* heuristic from [12], which replaces the random decisions of the verifier by a hash function over the parameters of the proof. As this requires no trust in the confidentiality of a setup this is called *transparent* setup instead [6]. This function is used instead to make the choices that the verifier would do randomly. The hashfunction is not selected by the prover, but it is part of the common shared data. [12, 17, 18]
Although there is some controversy about it, this heuristic is secure for honest verifier ZKPs according to [18] in the *Random Oracle Model*.

2.4. Proof of Knowledge

Many ZKPs[10] are ZKPs *of knowledge*. Proof of knowledge [19] intuitively means that the prover proves its knowledge about a witness usable to prove the given problem in polynomial time. Therefore some (explicit or extractable) knowledge is guaranteed to be possessed by the prover. This can be formalized by defining another probabilistic polynomial algorithm extracting the witness using the prover as an oracle machine. [19]

2.5. Example: Blackbox Workflow of a ZKP

The properties mentioned before are about the requirements a ZKP should fullfill, not about how one would use a ZKP in a concrete way. As even simple examples of ZKPs are too long for this paper, this is now discussed from a blackbox perspective and assumes that eventual setups for NZKPs are already done. In Figure 2 this high-level view is depicted for non-interactive ZKPs and to its parts are now referred to with (1) to (8). At first (1) the statement over the common input is transformed into the input type usable by the zero knowledge proof (2), in most cases this is some low level representation like boolean or arithmetic circuits. These are circuits over finite fields. For the circuits the satisfiability problem[11] is to be proven. There are programs to transform from higher level languages like TinyRAM (a small subset of the programming language C) into this input format [4]. The resulting number of gates $|C|$ of the circuit C is one variable of the proof length and costs as shown in Section 3. This problem representation is then used by the prover

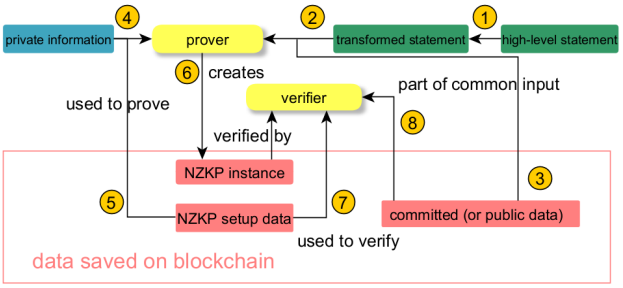


Figure 2: A high level view of the usage of non-interactive ZKPs.

(6) to either do an interactive proof or to generate the non-interactive proof over the common input (3) and its private information (4) using the proving key of the setup data (5). The verifier checks this (non-)interactively using (the common verification key (7) of the setup data and) the input data of the proof (8). [4, 6, 7].
Note that for statements like that a transaction on a blockchain [12] [20] is correct, data (either public or committed[13]) of that blockchain is part of the common input of the proof, so the proof refers to that specific blockchain. The user can formulate the statements on a higher level language and let the ZKP system do the complex mathematical part. Even without deep mathematical understanding of ZKPs usage of a provided implementations is possible. For such implementations see [4].

3. Examples of Modern ZKPs

In this section ZKPs are compared regarding their capabilities. This is a limited and simplified selection due to the complexity and extent of the topic. For this the popular term *zk-SNARKs* is introduced in Subsection 3.1. Then newer ZKPs called *zk-STARKs* are covered in Subsection 3.2. The popular ZKP *Bulletproof* and *Ligero* as zk-SNARK without and *Sonic* as a zk-Snark with trusted setup are discussed in the remaining Subsections.

3.1. zk-SNARKs

Zero Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARKs) is a common term for NZKPs. After Section 2 only the word *succinct* has to be explained, which means that the proof size and verification time are less than linear in the input size or even constant. Some zk-SNARKs use trusted setups like Sonic [22], while more recently zk-SNARKs without trusted setup (transparent setup or also interactively useable) like Ligero [23] have been created. [4, 6, 24]

3.2. zk-STARKs

Zero Knowledge Scalable Transparent Arguments of Knowledge (zk-STARKs) use cryptographic hashfunctions

9. This group does not have to consist of everyone that ever uses the system. It can be a reasonably sized subset balancing the trustworthiness of the setup and setup costs.
10. including the ones presented in Section 3
11. Showing that there exists inputs to the circuit so that the specified output (e.g. 1 for boolean circuits) is achieved.
12. or a similar system like central signature authority [12], as long as provers and verifiers trust the integrity of the data provided by it
13. Committing data means to use a one-way hashfunction so that the given argument cannot be changed without altering the hash, but the hash also contains no useable information about the used argument [21].

doi: 10.2313/NET-NET-2022-07-1_08

for their security assumptions. Quantum computers cannot find collisions for these hashfunctions efficiently [25]. For this reason zk-STARKs are believed to be more resistant to quantum computers than other ZKPs. For non-interactive proofs they use the Fiat-Shamir heuristic. *Scalability* means quasilinear proving time ($n \cdot polylog(n)$) and polylogarithmic verification time. In [26, 27] zk-STARKs are defined and some are constructed. Their new feature is the combination of scalability on both the provers and verifiers side combined with their transparency. Thus their prover timer outperforms zk-SNARKs (or Bulletproof) [26, Section 1.3]. But the proof sizes of STARKs are significantly larger for small input sizes (like validating blockchain transactions), which make them less suitable for this common use case on blockchains. [24, 26, 27]

3.3. Bulletproof

Bulletproof [28] is a ZKP that is used to prove that a committed secret value $v \in Z_p$ is in a given range $[0, 2^n - 1]$. Bulletproof relies on the *Discrete Logarithm Problem* as security assumption. Bulletproof has the feature to accumulate m range proofs into one bigger range proof with a smaller total size. The size of the proof[14] consists of $2(log_2(n) + log_2(m) + 4)$ elements of G and always 5 elements of Z_p. Also a multi party protocol was proposed [28, section 4.5] that can be used for merging multiple proofs of multiple parties while each party keeps their secret with linear communication in the number of proofs m and a logarithmic number of rounds in n. Therefore the total size of the accumulated proof grows logarithmic per additionally range proof over $[0, 2^n - 1]$. More generally Bulletproof can be used to prove that a given computation over an arithmetic circuit C is correct. The computation times needed for prover and verifier are each linear in n respectively $|C|$ [4, table 3]. Also the non-interactivity is achieved using the Fiat-Shamir heuristic, so Bulletproof requires no trusted setup. [28]

3.4. Ligero

Ligero [23] is a zk-SNARK made non-interactive with a transparent setup using the Fiat-Shamir transform. Like zk-STARKs its security assumption is based on cryptographic hash functions. It is used to verify for a given arithmetic circuit C, which checks the membership of a language L in NP, whether an input x is in L. The sublinear proof size is in $\Theta(\sqrt{|C|})$, therefore Ligero is called a *succinct* non-interactive argument of knowledge (zk-SNARK). Both the running times of the verifier and prover are in $O(|C| \cdot log(C))$ [4, table 3]. This protocol can also be used with a multi-party protocol merging multiple instances for a better amortized run time for the verification. [23]

14. The proof size is the length of the communication between prover and verifier in the interactive case, otherwise the length of the one-way message.

3.5. Sonic

Sonic is a zk-SNARK with trusted setup. The proof size is constant in regard to the input. The proving time is in $|C| \log |C|$ as well as the costs of the universal updatable trusted setup. Updatable means that the trusted setup can continue indefinitely, i.e. new participants can be later added to increase the trust in the setup. The universal setup has not to be repeated for different circuits, instead all circuits with circuit sizes up to a at the setup fixed size are allowed. The verification time is linear to the size N of the inputs of the gates and also logarithmic in $|C|$ [4, Table 3]. The security bases on the *Algebraic Group Model*. [22]

3.6. Comparison

In Table 1 are the asymptotic runtimes and proof sizes of the mentioned ZKPs depicted. Be aware that for small input sizes zk-STARKs create proofs of significantly larger sizes than Bulletproof or zk-SNARKs (including Sonic and Ligero). Because zk-SNARKs were introduced several years earlier than zk-STARKs, they have more available implementations to use. As shown these algorithms have different strengths and should be chosen depending on the use-case. Note that this is only a limited selection of the many ZKPs, more are listed in [4].

4. Use Cases

It was shown in [29] and [30] that every problem in NP[15] has computational (non-)interactive ZKPs. Also the ZKPs in 3 all support at least all problems in NP as input. Of this wide applicability of ZKPs some potential use cases are discussed next.
Generally, there are a lot of applications of ZKPs on blockchains like anonymous payments, voting, age verification, risk assessment, or auctions [7, sec. Zero-Knowledge Proof Applications], smart contracts, verifying computations or delegated computing [4].

4.1. Signatures

Like shown in [12] one can create non-interactive signatures having a zero knowledge property, issued by a central authority that is not required for authentication.
There are interactive zero knowledge undeniable signatures like in [31]. They cannot be verified without the signer, making it much harder (or not possible) for anyone but the private key owner to convince a third party that a message is signed correctly. Besides proving the correctness of a signed message, they can also be used to prove (also a ZKP) that a message is not correctly signed to protect the signer against false accusations. [31]

15. Decision problems that can be solved in exponential time and checked with a witness in polynomial time

Name	Trusted Setup	Proof Size	Proving Time	Verification Time
Bulletproof	No	$O(\log M)$	$O(M)$	$O(M)$
Ligero	No	$O(\sqrt{\|C\|})$	$\|C\| \log \|C\|$	$\|C\| \log \|C\|$
STARK	No	$O((\log \|C\|)^2)$	$O(\|C\| \ (\log \|C\|)^2)$	$O(\|C\|)$
Sonic	Yes	$O(1)$	$\|C\| \log \|C\|$	$N + \log \|C\|$

TABLE 1: Table part from [4, Table 3], $|C|$ is the number of gates of the computation expressed as arithmetic circuit, M the number of *And* gates in it, N the length of the inputs and outputs of the computation [4, Table 3].

4.2. Private Transactions on Blockchains

Digital currencies like Bitcoin are pseudo-anonymous. The transactions between all addresses are publicly visible to verify the correctness of the transactions. If an identity is linked to an address, the transaction history belonging to that address is retraceable. Using a new address for each new transaction or coin mixers makes this harder, but does not implicitly guarantee anonymity. [32]
Digital currencies for decentralized and trustless payments like the protocol Zerocash described in [20], solve this problem with the use of zk-SNARKS. Such currencies thereby offer a possibility for more anonymous (hidden amount, origin and destination) payments. The payment is found by scanning over the block chain using the corresponding private key searching for a commitment to the related address. [20]
Furthermore, for such currencies a trusted setup might be a disadvantage. As this requires trust in the correctness of key ceremonies with a big number of participants like that of Zcash that are only insecure if all participants are dishonest [16]. So instead NZKPs with a transparent setup might solve this issue. For example the previously mentioned Bulletproof which requires no trusted setup is used by the crypto currency Monero to prove that the sum of committed inputs is greater than the committed outputs of a transaction. [28, 33].

4.3. Verifying Computations

Even for checking computations on a von Neumann RISC architecture like vnTinyRAM in [34] verifying the computations via generating arithmetic circuits is possible for moderate code lengths. In this scenario both client and server know a function F and a given input x, the server knows or computes a secret w so that $z = F(x, w)$. The zk-SNARK used enables verifying the correct computation while the server keeps its secret and also the verification needs only very limited ressources. The cited work tests up to 32.000 machine cycles and 10.000 instructions on a desktop computer, achieving universality of the computations as only the time bound of the program execution has to be known ahead of the proof. But the work also shows that ZKPs for universal computations are still expensive for bigger and more complex programs. [34]

5. Related Works

A survey paper about ZPKs and the more recent development is [35]. An extensive document about the concepts of ZKPs is [6]. For a basic overview of some of the theory of interactive ZKPs see [3]. A basic example of a ZKP can be found in [29, Protocol 4]. [36] gives an good listing of some the theory of NIZKPs and the research history. [37] is a short overview of the use of NIZKPs in Blockchains. The later mentioned paper about Bulletproof [28] is also possibility to get a better understanding of a modern NIZKPs. For an overview of some of the many use cases and implementations of non interactive zero knowledge proofs for blockchains see [4].

6. Conclusion

As shown ZKPs have a complex theory and wide applicability. The aim of these algorithms is to minimize released information that is not to be proven, but is needed to prove something. The basic zero knowledge, soundness and completeness properties should definitely be fulfilled by all ZKPs. Especially the zero knowledge property shows a way to formalize that no usefull information except the validity of the statement to be proven is released. ZKPs are an interesting topic and wide research field with much theory and an important part of cryptography. The use of this technology on blockchains is a promising ongoing development as well as the improving NZKPs with transparent setup removing the trust issues of trusted setups. Also its many use cases like signatures, crypto currency transactions or even verifying computations are very versatile.

References

[1] C. Jackson, S. Russell, and S. Sons, *Security from First Principles*. Sebastopol, CA, USA: O'Reilly Media, Inc. [Online]. Available: https://www.oreilly.com/library/view/security-from-first/9781491996911/ch04.html

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, Feb. 1989. [Online]. Available: https://doi.org/10.1137/0218012

[3] O. Goldreich, "Zero-knowledge twenty years after its invention," 01 2003.

[4] J. Partala, T. Nguyen, and S. Pirttikangas, "Non-interactive zero-knowledge for blockchain: A survey," *IEEE Access*, vol. PP, pp. 1–1, 12 2020.

[5] B. J. Copeland, "The Church-Turing Thesis," Jan 1997, [Online; accessed 5. Dec. 2021]. [Online]. Available: https://plato.stanford.edu/entries/church-turing

[6] ZKProof, "Zkproof community reference," December 2019.

[7] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, no. 4, pp. 198–205, 2021.

[8] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge," in *Advances in Cryptology — CRYPTO' 88*, S. Goldwasser, Ed. New York, NY: Springer New York, 1990, pp. 37–56.

[9] J. Thaler, "Proofs, arguments, and zero-knowledge," August 2021, [Online; accessed 5. Dec. 2021]. [Online]. Available: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

[10] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, vol. 7, no. 1, pp. 1–32, Dec. 1994. [Online]. Available: https://doi.org/10.1007/bf00195207

[11] U. Feige and A. Shamir, "Witness indistinguishable and witness hiding protocols," 1990.

[12] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*. Springer Berlin Heidelberg, 1987, pp. 186–194. [Online]. Available: https://doi.org/10.1007/3-540-47721-7_12

[13] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*. ACM Press, 1988. [Online]. Available: https://doi.org/10.1145/62212.62222

[14] R. Canetti and M. Fischlin, "Universally composable commitments," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 19–40.

[15] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 698–728.

[16] "Parameter Generation - Zcash," Aug 2019, [Online; accessed 13. Dec. 2021]. [Online]. Available: https://z.cash/technology/paramgen

[17] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs, "Fiat-shamir: from practice to theory," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. ACM, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3313276.3316380

[18] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *Advances in Cryptology — EUROCRYPT '96*, U. Maurer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 387–398.

[19] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 390–420.

[20] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 459–474.

[21] A. Jain, S. Krenn, K. Piertrzak, and A. Tentes, "Commitments and effcient zero-knowledge proofs from learning parity with noise," 2021.

[22] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings," Cryptology ePrint Archive, Report 2019/099, 2019, https://ia.cr/2019/099.

[23] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2017. [Online]. Available: https://doi.org/10.1145/3133956.3134104

[24] "Zero-Knowledge Proofs: STARKs vs SNARKs | ConsenSys," Dec 2021, [Online; accessed 11. Dec. 2021]. [Online]. Available: https://consensys.net/blog/blockchain-explained/zero-knowledge-proofs-starks-vs-snarks

[25] "Defeating Quantum Algorithms with Hash Functions," Feb 2017, [Online; accessed 25. Feb. 2022]. [Online]. Available: https://research.kudelskisecurity.com/2017/02/01/defeating-quantum-algorithms-with-hash-functions

[26] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Report 2018/046, 2018.

[27] ——, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology – CRYPTO 2019*. Springer International Publishing, 2019, pp. 701–732. [Online]. Available: https://doi.org/10.1007/978-3-030-26954-8_23

[28] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," Cryptology ePrint Archive, Report 2017/1066, 2017, https://ia.cr/2017/1066.

[29] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *Journal of the ACM*, vol. 38, no. 3, pp. 690–728, Jul. 1991. [Online]. Available: https://doi.org/10.1145/116825.116852

[30] M. Blum, A. de Santis, S. Micali, and G. Persiano, "Noninteractive zero-knowledge," pp. 1084 – 1118, 1991.

[31] D. Chaum, "Zero-knowledge undeniable signatures (extended abstract)," in *Advances in Cryptology — EUROCRYPT '90*, I. B. Damgård, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 458–464.

[32] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, 2011, pp. 1318–1326.

[33] "Moneropedia: Bulletproofs," Dec 2021, [Online; accessed 13. Dec. 2021]. [Online]. Available: https://web.getmonero.org/resources/moneropedia/bulletproofs.html

[34] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct Non-Interactive zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[35] S. Kassaras and L. A. Maglaras, "Zkps: Does this make the cut? recent advances and success of zero-knowledge security protocols," *CoRR*, vol. abs/2006.09990, 2020. [Online]. Available: https://arxiv.org/abs/2006.09990

[36] H. Wu and F. Wang, "A survey of noninteractive zero knowledge proof system and its applications," *TheScientificWorldJournal*, vol. 2014, p. 560484, 05 2014.

[37] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, no. 4, pp. 198–205, 2021.

SCTP: Are you still there?

Zeynep Ince, Richard von Seck*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: zeynep.ince@tum.de, seck@net.in.tum.de

***Abstract*—Stream Control Transmission Protocol (SCTP) is a transport layer protocol initially designed for telephony signalling over IP networks. It is similar to Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Moreover, it has some crucial differences making the protocol outstanding such as multihoming and multistreaming. Through these features, multiple streams can be handled simultaneously, and the transmission goes on in case of a failure without interruption. So why is SCTP less used and known compared to TCP and UDP? This paper will try to answer this question in more detail by comparing SCTP with other transport protocols and detailing the current state-of-art of SCTP research.**

***Index Terms*—Stream Control Transmission Protocol, multihoming, multistreaming, Transmission Control Protocol, User Datagram Protocol**

1. Introduction

Reliable communication has been provided by Transmission Control Protocol (TCP) and unreliable communication has been provided by User Datagram Protocol (UDP) for many years [1]. However, because of the limitations they imposed, a third protocol named Stream Control Transmission Protocol was introduced. SCTP is a connection-oriented, message-based communication protocol in the transport layer that can handle multiple simultaneous streams. In 2000 it was standardized by the Internet Engineering Task Force (IETF) and the starting point of SCTP was to transport Public Switched Telephone Network (PSTN) signalling messages over IP networks [2]. TCP and UDP are the underlying concepts that made SCTP possible in the first place by combining their best features [3]. All the similarities aside, SCTP has some distinct differences and these features are multihoming and multistreaming. However, SCTP is still less known and used compared to TCP and UDP. In this paper, we will discuss the reason behind this situation by identifying the current state-of-the-art of SCTP research.

The remainder of this paper is organized as follows: First, we will take a brief look at the SCTP terminology in Section 2 and then compare SCTP with TCP and UDP in Section 3. Next, we will list some arguments about why it is less known and used in Section 4. In Section 5, we will provide an overview of the current use cases and actual specifications of SCTP and systems trying to optimize their performances using it. Finally, we will close with some conclusions and a brief overview in Section 6.

2. Brief Terminology of SCTP

Chunk: A unit of information that is sent within a packet.

Association: A protocol relationship between the two endpoints that can be uniquely identified by the transport addresses used by them; a broader concept than a connection [2].

Heartbeat: A type of chunk to check the availability of the idle destination addresses that are part of the built association. If the heartbeat acknowledgement is not returned, that particular IP address will be declared as "down" [4].

Stream: Unidirectional logical channel established from one endpoint towards another [3].

SACK: Selective Acknowledgment. When a message is received by one of the endpoints, the other endpoint should be notified back with a SACK [3]. A retransmission is only generated when SACKs report missing chunks [5].

Multihoming: Enables the SCTP host to establish an association with another host over multiple interfaces identified by different IP addresses [4].

Multistreaming: The capability to transmit several independent streams in parallel, meaning that each message sent to a data stream can have different final destinations [6].

2.1. Four-Way-Handshake

Since connections are initialized between unreliable hosts and over the unreliable internet communication system, a mechanism is needed to prevent errors [7]. The mechanism to establish an SCTP association is called four-way-handshake. The process takes place following these steps:

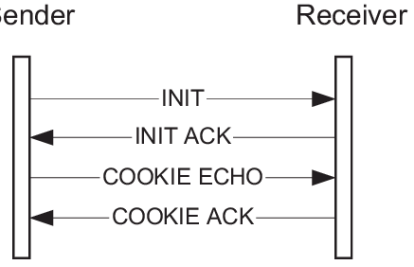


Figure 1: SCTP Four-Way-Handshake [8].

doi: 10.2313/NET-NET-2022-07-1_09

1) The client sends an INIT signal to the server to start an association.
2) Once the server receives the INIT signal, it sends back an INIT-ACK response to the client, which contains a state cookie. This state cookie consists of a timestamp, referring to the life span of it and a Message Authentication Code (MAC), which is created by the server, including a secret key that only the server knows.
3) When the INIT-ACK signal is received, the client sends a COOKIE-ECHO response to echo the state cookie.
4) Subsequently, the authenticity of the state cookie is verified by using the secret key the MAC encapsulates. Before sending the COOKIE-ACK response, the server allocates the resources and the association's state is now ESTABLISHED.

The server does not keep any state information until the very end and the full handshake should be completed in order to have an actual state maintained on the server and no resources are allocated until the COOKIE-ECHO message is received by the receiver [8]. Moreover, when an endpoint decides to perform a shutdown, the association on each one of them will stop accepting new data and only deliver data in queue before closing [2].

3. Comparison of Features

Instead of listing the features of all protocols individually, we start by comparing them to have an overview of how they are positioned towards each other. See Table 1 for a brief overview.

Service/Features	TCP	UDP	SCTP
Transmission	Byte-oriented	Message-oriented	Message-oriented
Connection management	Connection-oriented	Connectionless	Connection-oriented
Reliability	Reliable	Unreliable	Reliable
Data delivery	Strictly ordered	Unordered	Partially ordered
Multistreaming	No	No	Yes
Multihoming	No	No	Yes

TABLE 1: Comparison of Protocols

Byte-oriented: TCP does not stream bytes over the Internet, opposing what it can be understood from this term. Enough bytes from the sending process are buffered by the TCP on the source host in order to fill a packet. This packet is then sent to its peer on the destination post. TCP there empties the contents into a receive buffer [7].

Message-oriented: Transported sequences of messages are in groups of bytes. In SCTP, these groups are called "chunks", as mentioned in Section 2.

Connection-oriented: Before the data is being exchanged, a connection should be established. In TCP this process is called the Three-Way-Handshake and in SCTP, as mentioned in Section 2.1, it is called the Four-Way-Handshake.

Connectionless: The transmission from the source to the destination starts right away, without verifying the state of the server.

3.1. TCP and UDP

TCP was standardized in 1981 and it has been the most widely chosen option for transmitting data ever since [9]. It is a connection-oriented, reliable protocol that guarantees none of the transmitted packets will get lost: TCP can retransmit them. In this sense, packets are sent strictly ordered and the receiver collects and reorders these segments conveniently. When a failover situation occurs where the data could not arrive in order, the TCP stack will wait for the retransmission and others are held. This is the situation that we call HOL blocking (Head of Line blocking) because of the strict order-of-transmission delivery of data [1].

On the other side, UDP is a connectionless protocol and packets do not necessarily arrive in order. The connection establishment is not checked like in SCTP and TCP. Packets can quickly go missing and the sender will not know whether the transmission is completed either, making UDP an unreliable protocol. The head-of-line blocking problem does not occur in UDP as well. Furthermore, TCP is byte-oriented while UDP is message-oriented.

3.2. UDP and SCTP

UDP is a connectionless, unreliable transport protocol contrary to SCTP. However, they are both message-oriented. In SCTP, an association is established after a four-way-handshake, but in UDP, no such process is needed. In UDP, the transmission starts right away without checking if it is received or not and there is no retransmission process either. It is helpful for cases where we need live real-time connections and retransmission is unnecessary. For example, retransmitting the position of an online game character from 5 seconds ago is not logical since it is not valid anymore. On the other side, SCTP can detect the loss of a packet rapidly in favor of SACK usage. Moreover, the UDP Header is much smaller when compared with SCTP and TCP; this makes UDP lighter and consequently attractive for fast and efficient handling of audio, image and video data traffic [10].

3.3. TCP and SCTP

TCP and SCTP are very similar protocols, and both of them are reliable and connection-oriented. SCTP was designed to push the edge of the envelope of TCP. The useful features of TCP were inherited and new features were added [11].

TCP uses a three-way-handshake where initial sequence numbers are being exchanged and SCTP uses a four-way-handshake including a signed cookie, as described in Section 2.1. However, the SCTP connection process is more complex [11] and the use of a cookie improves the vulnerability of TCP to SYN flooding [2]. Correspondingly, the security of the protocol is improved. [8].

SCTP is message-oriented, unlike TCP, which is byte-oriented. However, SCTP transmits its packets in chunks and TCP buffers enough bytes to fill a packet before sending, as described at the beginning of this section.

The two most important differences of SCTP compared to TCP are the multihoming and multistreaming features. In a traditional TCP connection, an IP address and a port is chosen from each end and with them, packets are sent and received. However, multihoming allows an SCTP association over multiple interfaces. Between the

different path options, a primary path is selected and the availability of the rest is constantly checked with heartbeats. If a failover situation occurs with the primary path, one of the other options is used and the transmission will go on without interruption [6]. Thus, the loss of a message does not affect the rest of the deliveries and the head-of-line blocking problem is avoided in SCTP. By means of this, the fault tolerance [12] and the robustness of the server is improved [8]. That is why it is a desired functionality [6].

Moreover, TCP has a strict order-of-transmission delivery mechanism [4] and SCTP has reliable ordered delivery and reliable unordered delivery services at the same time due to the multistreaming feature. This feature makes the transmission of several independent data streams in parallel possible. [8]. SCTP determines in which order to present the messages to the destination and this approach eliminates the head-of-line-blocking delay [4] that is caused by the strictly ordered delivery [1]. This combination of streams and unordered delivery simultaneously is helpful for Internet applications to have a better performance when a failover occurs, such as a network loss. By courtesy of this the overall latency is reduced and transmission efficiency is improved [3]. These features make SCTP valuable for real-time data transfers such as audio and video.

Taking into account all of these, we can say that SCTP is more robust and secure than TCP, under the assumption that the introduced features are used [13].

4. Why SCTP is less used

Even though all the advantages SCTP has over TCP, it is less known and used. The main reason behind this is that the client and server applications might need modifications such as upgrading IP stacks to use SCTP instead of TCP or UDP [9] and this would be too much work. In-network devices like NAT gateways, does not support SCTP well [14]. Moreover, TCP was first-to-market and for the most part, TCP is sufficient and works just fine. That is why TCP was the dominant transport protocol for a very long time [9].

In 2009, Google introduced a new transport protocol called QUIC: Quick UDP Internet Connections. It is a multiplexed low-latency transport protocol designed to improve the web performance [15]. As the Internet traffic increases rapidly, it is necessary to look for new technologies [16]. The most crucial advantage QUIC has over SCTP is that QUIC does not require changes to the operating system and this makes QUIC easily deployable with applications that are already in use. Briefly, QUIC can handle multiple request/response pairs concurrently on a single connection by using multiple streams and a packet loss does not block the rest of the connection [17], but it is still under active development and some specifications are still missing. Considering the apparent dominance of Google over the Internet, QUIC is widely used in Chrome clients [14]. This helped SCTP remain its obscurity too.

5. Current and Possible Use Cases

Still, SCTP has some essential use cases and it helps some systems optimize their performances. Multipath transport layer protocols such as SCTP are gaining increased attention every passing day [18]. SCTP is recently supported by a variety of operating systems, such as AIX, Solaris, Linux and Windows: Microsoft provides user space for SCTP implementations in the Windows family too [19].

5.1. Long-Term Evolution (LTE)

Diameter Protocol in LTE provides authentication, authorization, and accounting (AAA) services. LTE is closely related to 4^{th} generation mobile data transfer, which gives cause for higher data transmission. The multistreaming feature of SCTP is helpful at this point, making SCTP the up-front transport protocol being used to transport messages. Mobile consumers expect high-quality data experiences and invisible high-speed access.

5.2. Concurrent Multipath Transfer (CMT)

CMT is a process of using multiple networks to transfer data instead of selecting a single network interface for transmission. Several pieces of research have been done to test the multiple-path transmission of SCTP with CMT [19]. However, considering the dominance of TCP as a transport protocol, a proxy technique is needed to translate the TCP flows into SCTP streams without being obligated to make significant changes at end hosts or servers. Tachibana et al. [9] claim that the multihoming feature of SCTP would increase the aggregated throughput and the robustness of communication.

Liao et al. [5] introduced a modification of SCTP called cmpSCTP. With this solution, the transmission is updated based on real-time and all of the available paths are used simultaneously, unlike SCTP, where a chosen primary path is used. Moreover, as the states of paths change, the transmission strategy is also changed by cmpSCTP, and the flows between paths can be switched smoothly. Cloud computing is one of the examples where CMT and SCTP are used together by combining the multihoming feature of SCTP and multipath transfer technology of CMT [19].

5.3. Internet of Things (IoT) Sensors

The IoT is a giant network with connected devices and these devices share the data they collect over their sensors to help us understand and measure the planet around us. Sensors are embedded in most physical devices such as our smartphones and generate large amounts of real-time data. These are collected in sink nodes and transmitted over heterogeneous networks afterwards. It is essential that the packet loss rate is as low as possible and transmission quality is high. For this reason, the transport layer protocol should be chosen wisely. A switch of TCP and SCTP might be suitable for this case. Sun et al. [11] compared the performances of both protocols and proposed that SCTP is reliable, but TCP has higher transmission stability, which brought the idea of combining only the better sides of both protocols as a method out. This rendered the network's state prediction possible considering the packet loss rate. By courtesy of the multistreaming feature, multiple requests can be processed and

with the multihoming feature the transmission efficiency was improved. Considering both of these features belong to SCTP, it can also be used in the field of IoT.

5.4. Session Initiation Protocol (SIP)

SIP is a protocol for managing communication sessions such as voice and video calls over internet telephony or mobile phone calls over LTE. Large amounts of data are exchanged between SIP entities and the protocol is independent of the underlying transport protocol. It can be used with TCP, UDP or SCTP. However, choosing SCTP as the transport protocol would provide some crucial advantages [12].

As we have mentioned before, SCTP uses Selective Acknowledgement (SACK) to generate the retransmission of a missing chunk and retransmissions take place only after SACKs report them. The loss of a SIP message is detected immediately. Moreover, this loss does not affect the rest of the transmissions, so if multiple transactions are happening at the same time SCTP will handle them with relative ease. SIP entities choose the server on the next-hop by checking if it supports SCTP to establish an association [12].

5.5. Satellites

Satellite links ensure some essential services we use every day, such as navigation services, television and telephony and without them, the Internet may not be the same. Satellite networks have a large transmission distance. Therefore, problems like corruption losses due to wireless links and long propagation delays come into sight. However, TCP was not designed for such networks [20].

SCTP is recommended for running over satellite networks because of many reasons. In a satellite environment, multiple segment losses are incidental and with SACKs, it is possible to react rapidly. The multihoming feature derives satellite networks to be fault-tolerant and reliable and the multistreaming feature eliminates the HOL blocking by reducing the receiver buffer size requirements [18].

5.6. Datagram Transport Layer Security (DTLS)

A datagram provides a connectionless communication service across packet-switched networks and DTLS is a communications protocol that maintains security to datagram-based applications. In this sense, using DTLS over SCTP means providing a secure channel to applications that are using SCTP as their transport protocol. By courtesy of this, eavesdropping is prevented, where information gets stolen while being transmitted—this way the confidentiality, authenticity and integrity of the network is ensured. Using DTLS over SCTP reinforces preservation of message boundaries, ordered and unordered delivery of SCTP user messages and a large number of unidirectional and bidirectional streams [21].

6. Conclusion

This paper has compared SCTP with other transport protocols and listed the current use cases. SCTP is a message-oriented, connection-oriented and reliable protocol and most importantly it can deal with multihomed hosts and manage multiple streams at the same time. However, TCP has been the dominant protocol for many years and SCTP was not as known and used compared to the other protocols. The main reason for that is TCP being first to the market. There are fewer client and server applications supporting SCTP, but many applications support TCP. Furthermore, TCP is sufficient in most cases. The introduction of QUIC by Google that is easily deployable without any changes and is similar to SCTP, let the protocol remain unknown.

SCTP is currently being used in LTE technology, Concurrent Multipath Transmission, IoT sensors, Session Initiation Protocols, satellites, Datagram Transport Layer Security and cloud computing. SCTP has a usage area in the telecommunications industry that it is sufficient for, but it might be the case that QUIC will be preferred for other industries in the future. It is similar to SCTP and will keep getting better since it is still under active development.

References

[1] R. Stewart, M. Tüxen, and P. Lei, "Sctp: What is it, and how to use it?" in *Proceedings of BSDCan: The Technical BSD Conference*. Citeseer, 2008.

[2] R. R. Stewart, "Stream Control Transmission Protocol," RFC 4960, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4960.txt

[3] "Sctp and diameter parameters for high availability in lte roaming," Ph.D. dissertation, accessed: 17.12.2021. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-163254

[4] R. Stewart and C. Metz, "Sctp: new transport protocol for tcp/ip," *IEEE Internet Computing*, vol. 5, no. 6, pp. 64–69, 2001.

[5] J. Liao, J. Wang, and X. Zhu, "cmpsctp: An extension of sctp to support concurrent multi-path transfer," in *2008 IEEE International Conference on Communications*. IEEE, 2008, pp. 5762–5766.

[6] S. Y. Shahdad, G. Amin, and P. Sarao, "Multihoming and multi-stream protocol in computer networks," 2014.

[7] J. Postel *et al.*, "Transmission control protocol," 1981.

[8] G. Camarillo, H. Schulzrinne, and R. Kantola, "Signalling transport protocols," 03 2002.

[9] A. Tachibana, Y. Yoshida, M. Shibuya, and T. Hasegawa, "Implementation of a proxy-based cmt-sctp scheme for android smartphones," in *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2014, pp. 660–665.

[10] H. Mohamad Tahir, M. A. Abu Seman, S. Shelen, M. S. Selan, S. H. Abdi *et al.*, "Performance comparison of sctp and udp over mobile ad hoc networks," *International Journal of Computer Science Issues*, vol. 9, no. 4, pp. 443–448, 2012.

[11] W. Sun, S. Yu, Y. Xing, and Z. Qin, "Parallel transmission of distributed sensor based on sctp and tcp for heterogeneous wireless networks in iot," *Sensors*, vol. 19, no. 9, p. 2005, 2019.

[12] J. Rosenberg, H. Schulzrinne, and G. Camarillo, "The stream control transmission protocol (sctp) as a transport for the session initiation protocol (sip)," *Internet Engineering Task Force, Tech. Rep. RFC*, vol. 4168, 2005.

[13] R. R. Stewart and Q. Xie, "Stream control transmission protocol (sctp): a reference guide," 2001.

[14] A. Joseph, T. Li, Z. He, Y. Cui, and L. Zhang, "A Comparison between SCTP and QUIC," Internet Engineering Task Force, Internet-Draft draft-joseph-quic-comparison-quic-sctp-00, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-joseph-quic-comparison-quic-sctp-00

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

[16] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is quic?" in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[17] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath quic: A deployable multipath transport protocol," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[18] J. Deutschmann, K.-S. Hielscher, T. Keil, and R. German, "Multipath communication over terrestrial and satellite links," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2018, pp. 119–121.

[19] L. Zheng, X. Zhang, S. Zhang, and X. Chen, "Research on multipath network in cloud computing based on sctp," in *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2021, pp. 30–35.

[20] S. Fu, M. Atiquzzaman, and W. Ivancic, "Sctp over satellite networks," in *2002 14th International Conference on Ion Implantation Technology Proceedings (IEEE Cat. No. 02EX505)*. IEEE, 2003, pp. 112–116.

[21] M. Tuexen, R. Seggelmann, and E. Rescorla, "Datagram transport layer security (dtls) for stream control transmission protocol (sctp)," *Request for Comments*, vol. 6083, 2011.

Comparison of Different QUIC Implementations

Michael Kutter, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: michael.kutter@tum.de, jaeger@net.in.tum.de

***Abstract*—While the QUIC protocol was finalized by the IETF back in May 2021, the standard still leaves some design choices up to the developer. Especially for features like congestion and flow control, multiple streams, retransmission, packet size and 0-RTT, different approaches need to be considered. We give an overview of some of the considerations done by the developer and evaluate the performance of some implementations. We argue that future work needs to analyze the effect of the design choices on performance more in detail to find out which choice works best.**

***Index Terms*—QUIC, implementation, design choices, performance**

1. Introduction

The QUIC protocol specifications were finalized on May 2021 after nearly five years of development [1]. It is built on top of UDP, which enables support for middleboxes, as no new transport layer protocol is defined. The goal of this protocol was to improve performance for HTTPS connections, while also achieving high security [2]. This is realized in multiple ways. QUIC exchanges cryptographic information during the connection establishment, thus reducing the round-trip times (RTT) and amount of packets during the initial handshake (1-RTT). When reconnecting to a server, it utilizes the already share keys to directly send data during the handshake (0-RTT). It uses connection IDs to identify connections after an IP address changed, thus allowing immediate reconnection to the server. To avoid the head-of-line blocking problem, QUIC uses multiple independent data streams.

During the five years of developing the specifications, different implementations have evolved. In this paper, we compare QUIC implementations and outline the different approaches they use. We focus on features which are up to the developer, like congestion and flow control, multiple streams, retransmission, packet size and 0-RTT [1] based on the study done by R. Marx et. al in [3]. We also try to compare the performance of some QUIC and TCP implementations based on the test results of the paper by A. Yu and T. A. Benson in [4].

In chapter 2 we list all the implementations we choose for this analysis. Chapter 3 then outlines all the different design choices considered by the developers. Afterwards in chapter 4, we evaluate the performance.

TABLE 1: QUIC implementations

Name	Developer	Language	Version
aioquic [5]	Jeremy Laine	Python	RFC 9000
lsquic [6]	LiteSpeed Technologies	C	RFC 9000
ngtcp2 [7]	Tatsuhiro Tsujikawa	C	RFC 9000
quic-go [8]	Lucas Clemente et. al	Go	RFC 9000
mvfst [9]	Facebook Inc.	C++	draft-29
picoquic [10]	Private Octopus Inc.	C	draft-34

2. List of Implementations

The QUIC implementations taken for analysis are shown in table 1.

3. Design Choices

When implementing the QUIC standard, different design choices can be considered. In the following sections we outline design choices made by the listed implementations.

3.1. Congestion Control

Sending packets as fast as possible can lead to overloading the network and result in routers dropping packets. These packets then need to be retransmitted, which leads to a longer transmission time. To avoid this, congestion control algorithms are used. These algorithms limit the number of inflight packets, by controlling the congestion window.

The QUIC standard defined by the IETF provides an exemplary congestion control algorithm which is similar to the TCP New Reno algorithm [11]. Therefore, it is up to the implementation side to choose the algorithm. The most used algorithms are New Reno, CUBIC and BBR. Compared to the implementation for TCP they slightly differ but the concepts stay the same.

New Reno: This algorithm is based on the Reno algorithm but improves during retransmission [12]. It begins with a "slow-start" phase, where it increases the congestion window by one for each acknowledged packet, resulting in exponential growth. Once multiple duplicate ACKs were received, or a packet was not acknowledged (retransmission timeout), it enters the "fast-recovery" phase. During this phase, it immediately retransmits the lost segments. When the retransmission was fully acknowledged it keeps the current congestion window. But if the retransmission was only partially acknowledged, it halves

doi: 10.2313/NET-NET-2022-07-1_10

the current congestion window. After that it exits the "fast-recovery" phase and linearly increases the congestion window until the next packet was lost where it enters the "fast-recovery" phase again. This algorithm is a loss-based algorithm.

CUBIC: This algorithm is also a loss-based algorithm and also similar to the Reno algorithm [13]. It starts with the same "slow-start" phase. When a segment is lost it also enters the "fast-recovery" phase, where it retransmits the lost segments and halves the congestion window. The main difference is after the "fast-recovery" phase. Here, it uses a cubic function to increase the congestion window. In the beginning it increases the congestion window very slowly but increases it very fast later on. Compared to the Reno algorithm, it recovers faster from packet loss while not running into the next packet loss immediately. This algorithm is also the default congestion control algorithm in the Linux kernel for TCP.

BBR: This algorithm is called Bottleneck Bandwidth and Round-trip propagation time (BBR) and is a congestion-based algorithm developed by Google [14]. Compared to loss-based algorithm, it handles congestion based on the round-trip time. The algorithm tries to operate at the optimal point which is defined by the Bandwidth Delay Product $BDP = bandwidth \cdot RTT$. However, it is not possible to measure the bandwidth and the RTT at the same time, therefore estimated values are used [14]. The main advantage of this algorithm is that it does not fill the buffer of intermediate network nodes because this would lead to a bigger RTT.

Of the analyzed implementation New Reno is implemented by aioquic, ngtcp2, quic-go, mvfst and picoquic. CUBIC is implemented by lsquic, ngtcp2, quic-go mvfst and picoquic. BBR is implemented by lsquic, ngtcp2 mvfst and picoquic. We can see that a lot of implementations leave it to the user which algorithm to choose. This is escpecially beneficial for networks using different congestion control algorithms because some algorithms might outperform other algorithms which can lead to a sender barely sending any data due to a small congestion window [13].

3.2. Flow Control

While congestion control is about preventing the network from being overloaded, flow control is responsible for not overloading the receiver buffer. This is needed because the application might not be able to read data in the same speed the network delivers it, or the data was not received in the correct order. In TCP, each ACK packet provides the current receive window, which indicates the current available space in the receiver buffer [15]. Compared to TCP, QUIC allows multiple parallel data streams, therefore the QUIC protocol additionally applies flow control for each stream. The abstraction between stream level and connection level flow control is needed to prevent a single stream consuming the entire receivers buffer. This limitation is done through the MAX_STREAM_DATA (stream level) and the MAX_DATA (connection level) parameters [1]. Here, multiple approaches are possible to implement, and the following are most common.

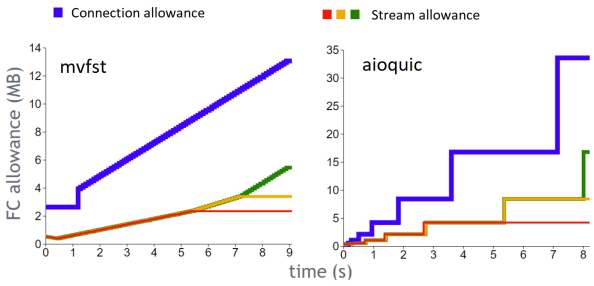


Figure 1: Flow control of mvfst and aioquic [3]

Static Allowance: With static allowance, the receive buffer has a fixed size, while the maximum allowance rate is increased linearly [3]. Typically, when the application has handled 50% of the received data in the buffer, the receive window is updated by adding the current buffer size. The downside of this method is that it can cause the sender to stop sending, when the updates of the receive window are delayed. This method is used by most of the analyzed implementations (lsquic, ngtcp2, quic-go, mvfst, picoquic), as it is easy to implement.

Growing Allowance: Growing allowance works similar to the static allowance method, but allows the receiver buffer to grow over time. This reduces the problem that this sender may stop sending, due to delayed receive window updates. Of the analyzed implementations, only aioquic used this method.

A detailed example of these approaches can be found in Figure 1 with mvfst (static allowance) and aioquic (growing allowance).

Regardless of the above mentioned methods, the most important aspect in regards to performance, is the size and the frequency of the receive window updates, as too small receive windows or too few updates can lead to a stalling sender. Flow control in QUIC remains an open issue, therefore, further study is required to identify the best possible approach.

3.3. Multiple Streams

TCP offers a single reliable in-order stream to transmit data. When transmitting independent resources, this is vulnerable to head-of-line blocking. This occurs when a single resource prevents other resources from being received, which happens when TCP loses a packet. The QUIC protocol defines multiple data streams to get around this problem [1]. In order to handle and multiplex these streams, a scheduler is required. Therefore, two approaches can be used to divide the bandwidth between the resources.

Sequential: When using a sequential scheduler, all data of one stream is send first before sending data of another stream. This is expected to work best for loading Web pages, as the application can prioritize which data should be sent first. However, this can lead to head-of-line blocking again e. g. when the data of a single stream is too big. Aioquic, ngtcp2 and picoquic are using this scheduler approach.

Round-Robin: When using a Round-Robin scheduler, the bandwidth is equally distributed between all resources.

doi: 10.2313/NET-NET-2022-07-1_10

However it is important to avoid sending data of multiple different streams inside one packet because this could lead to head-of-line blocking again. The typical approach is to send a few packets of data of the same stream before switching to another stream. The downside of this approach is that it can take longer to receive all data of a single stream. lsquic, quic-go and mvfst are using this approach.

We can see that stream multiplexing can have a significant impact on performance. Therefore, the QUIC standard additionally requires the implementations to have a prioritization system, which the application can use to prioritize streams [1]. With this system, the impact of the scheduler approach becomes less important, as higher priority streams will be sent first. However, this is only relevant if the application supports prioritization of resources. We think that the round-robin scheduler is a more general approach because it avoids head-of-line blocking regardless of the data size and it could also simulate a sequential scheduler when using the prioritization system.

3.4. Packet Size

The QUIC protocol requires a minimum UDP payload size of 1200 bytes, but to further improve throughput, a bigger packet size is required [1]. When using bigger packets, the chance of dropping a packet, due to an intermediate network node not supporting this size, highly increases. Therefore, it is recommended to either use Path MTU Discovery (PMTUD) or Data Packetization Layer Path MTU Discovery (DPLMTUD)[1], to find out the maximum packet size supported by all network nodes [1]. These methods works by sending a large packet to the destination. If the corresponding ICMP error message is received, we know that the MTU was too large and and intermediate network node dropped the packet. Therefore, we repeat the process by reducing the packet size until successful transmission. This feature is currently only supported by lsquic, quic-go and picoquic. The other implementations use a fixed packet size. Therefore, they are not allow to exceed the minimum UDP payload of 1200 bytes to ensure compatibility of intermediate network nodes.

3.5. Client Validation of 0-RTT

A new feature in the QUIC protocol is the 0-RTT connection establishment, where it is possible to send data before receiving any response from the server. This is made possible by reusing the preshared encryption keys of the session ticket, which were negotiated in the first connection. This feature is vulnerable to replay attacks and amplification attacks. Therefore, the QUIC protocol specifies that the server is not allowed to send back more than three times the data it received from the client until the address of the client is validated [1]. The validation can be done in two ways.

Approach 1: After the client requested data during the 0-RTT connection establishment, the server answers directly within the 3x limit. It then waits until the acknowledgment of the client. When the acknowledgment was received the address can be considered validated and the server can send the rest of the data. The disadvantage of this method is that the server can only respond in the beginning with small packets inside the 3x limit. However, it is possible to increase this limit by adding some padding to the initial data request. This approach is currently used by aioquic and ngtcp2.

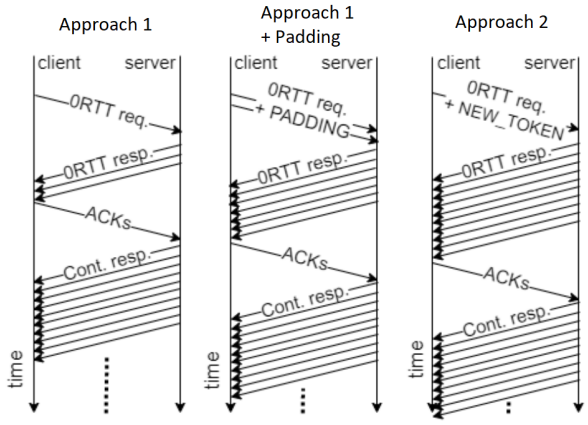


Figure 2: Client validation of 0-RTT [3]

Approach 2: During the first connection establishment (1-RTT), the server sends an encrypted NEW_TOKEN frame to the client, which can be used by the client during the 0-RTT connection establishment to validate its address. The server then can ignore the 3x limit and directly send large amounts of data to the client, if the token matches. This is done by lsquic, quic-go, mvfst and picoquic.

A detailed diagram about these approaches can be found in Figure 2.

4. Performance

A. Yu and T. A. Benson measured the performance of different QUIC implementations and compared them to different TCP implementations in their paper “Dissecting Performance of Production QUIC“ [4]. Compared to most other papers, they focused more on already deployed implementations, instead of testing it on a local setup. This approach in benchmarking resembles a more real world scenario. In their analysis, they also tried to diffentiate, if the results were due to the protocol specifications or due to the design of the implementation.

For their benchmarking, they use public available endpoints by Google, Facebook and Cloudflare on the server side. On the client side they choose to use cURL, Google Chrome, Facebook Proxygen and ngtcp2. All these implementations are using the HTTP/2 (H2) stack for TCP and HTTP/3 (H3) stack for QUIC. With the Network Link Conditioner, they simulate different network conditions. It is also worth mentioning, that they setup the flow control mechanism to not have any impact on the performance.

When transmitting a single resource, we and the authors excpect similar results between QUIC and TCP. For a small file size the QUIC implementations did outperform

1. https://blog.litespeedtech.com/2020/10/19/improve-performance-with-dplpmtud/

doi: 10.2313/NET-NET-2022-07-1_10

the TCP implementations. This is due to the improved handshake of the QUIC protocol. For larger files the impact of the improved handshake minimizes. Consequently, the performance between all the implementations are similar. However, when adding packet loss to the network, the Cloudflare H3 endpoint worsens compared to H2. The authors identified that this is due to different congestion control algorithms between H3 (CUBIC) and H2 (BBR). The other endpoint which stands out is Facebook. Their H3 endpoint performed significantly worse when adding extra delay. It was identified that this is due to a bug in the congestion control algorithm. We can see that the choice of the congestion control algorithm can have a huge impact on the performance as already outlined in chapter 3.1.

When transmitting multiple resources, we and the authors expect QUIC to perform better as TCP, due to QUIC's protocol design with the introduction of multiple data streams. However, the results were similar compared to transmitting a single resource. For small files, the H3 endpoints performed better, which could be traced back to QUIC's handshake design again, and for larger files the performance was similar. The only exception was the Cloudflare endpoint. Here, H3 also outperformed H2 for larger resources. The authors traced this issue back to different application configurations which favored the H3 implementation. The authors also analyzed the effect of the different scheduling approaches. Cloudflare is using a sequential scheduler, while Facebook and Google are using a round-robin scheduler. Here, the different scheduler did not have any effect on the performance due to the prioritization system by the H3 stack. This is also the same, which we concluded in section 3.3.

The authors concluded that most performance discrepancies are a result of the developers design or the operators configuration. These results can also be verfied by other papers [16], [17].

5. Conclusion and future work

In this work, we have discussed multiple design choices which needs to be considererd when implementing the QUIC protocol. We saw that there are multiple different approaches to implement congestion and flow control, multiple streams, packet size and client validation of 0-RTT. While not all aspects will have high impact on the performance and some might be application dependent, we concluded that further research is needed to find out which approach works best in practice.

We also analyzed the tests performed by A. Yu and T. A. Benson, where they compared the performance of different QUIC implementations with TCP over public available endpoints. We saw that in gerneral the QUIC implementations had the advantage when transmitting small resources due to the improved handshake design of the protocol. For larger files, the performance balances out because the impact of the handshake minimizes. However, we saw some discrepancies to this behavior. Most of these performance differences could be traced back to the developers design of the implementations or the configuration of the operators.

We feel that future analysis is needed to compare the performance of the different implementations. It is espacially important to focus on which design choices impacts the performance of the protocol.

References

[1] J. Iyengar and M. Thomson, "Rfc 9000," https://datatracker.ietf.org/doc/rfc9000/, May-2021, [Online; accessed 26-February-2022].

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Sweet, 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," August-2017.

[3] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity," August-2020.

[4] A. Yu and T. A. Benson, "Dissecting Performance of Production QUIC," April-2021.

[5] J. Laine, "aioquic," https://github.com/aiortc/aioquic, [Online; accessed 26-February-2022].

[6] L. Technologies, "Litespeed quic (lsquic) library," https://github.com/litespeedtech/lsquic, [Online; accessed 26-February-2022].

[7] T. Tsujikawa, "ngtcp2," https://github.com/ngtcp2/ngtcp2, [Online; accessed 26-February-2022].

[8] "A quic implementation in pure go," https://github.com/lucas-clemente/quic-go, [Online; accessed 26-February-2022].

[9] Facebook, "mvfst," https://github.com/facebookincubator/mvfst, [Online; accessed 26-February-2022].

[10] "picoquic," https://github.com/private-octopus/picoquic, [Online; accessed 26-February-2022].

[11] J. Iyengar and I. Swett, "Rfc 9002," https://datatracker.ietf.org/doc/rfc9002/, May-2021, [Online; accessed 26-February-2022].

[12] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "Rfc 6582," https://datatracker.ietf.org/doc/html/rfc6582, April-2021, [Online; accessed 26-February-2022].

[13] M. Geist and B. Jaeger, "Overview of TCP Congestion Control Algorithms," May-2019.

[14] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR Congestion-Based Congestion Control," Septemper-2016.

[15] I. S. I. U. of Southern California, "Rfc 793," https://datatracker.ietf.org/doc/html/rfc793, Septemper-1981, [Online; accessed 26-February-2022].

[16] S. Endres, J. Deutschmann, K.-S. Hielscher, and R. German, "Performance of QUIC Implementations Over Geostationary Satellite Links," Feburary-2022.

[17] M. Moulay, F. D. Munoz, and V. Mancuso, "On the Experimental Assessment of QUIC and Congestion Control Schemes in Cellular Networks," June-2021.

Survey on Machine Learning-based Autoscaling in Cloud Computing Environments

Oliver Lemke, Sayantini Majumdar*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: oliver.lemke@tum.de, sayantini.majumdar@tum.de

***Abstract*—Modern cloud computing systems have demonstrated great aptitude for providing accessible, cheap, and scalable computing infrastructure to businesses and the public at large. However, the computing model comes with a variety of challenges for cloud service providers. Especially the task of automated distribution of computing resources, called autoscaling, has proven difficult so solve. A variety of different approaches have been proposed, chief among them machine learning-based algorithms. Thus, this paper aims to give an overview of recent developments in the field of machine learning-based autoscaling. In particular, we compare and contrast two approaches: MLScale, a supervised learning-based solution utilizing neural networks and multiple linear regression, and RLPAS, employing an algorithm based on SARSA reinforcement learning. We come to the conclusion that RLPAS' ability to predict required resource spikes and provision resources proactively, puts it at a decisive advantage compared to the reactive MLScale. However, as RLPAS is much more algorithmically complex, we propose that further research is required to show safe and effective scaling for more complex, real-world problems.**

***Index Terms*—cloud computing, autoscaling, machine learning**

1. Introduction

Cloud computing (CC) is an architecture that enables on-demand network access to a pool of computing resources, such as networks, servers, storage, applications, or other services [1]. Hardware resources are owned and managed by a cloud service provider (CSP), allowing customers remote access. Besides eliminating capital expenditure for users, the consolidation of resources also reduces operating expenses due to higher resource utilization [2, Chapter 1]. Combined with the offer of constant availability and pay-as-you-go pricing options [3], this service model is thus considered an attractive option, especially for small- and medium-sized businesses [4] [5].

In order to reliably supply a service that can handle large variations in requested operations, e.g. due to sudden user demand, CSPs have to be able to automatically scale the resources distributed to a specific application. This allocation is a highly complex and important process, as both under- and oversupplying resources will result in major costs to the provider, in the form of contract violations and excess operating expenses respectively [6, Chapter 7.4]. Yet, major CSPs like Oracle still rely on manually set autoscalers which base scaling decisions purely on simple thresholds [7] that are unable to adapt adequately to fluctuating user demand.

We reason that machine learning (ML), a field which particularly excels in complex and dynamic environments, represents the most promising approach vector towards solving this problem. Thus, this paper aims to give an overview of recent advances, especially comparing MLScale [8] and RLPAS [9], proposals utilizing supervised (SL) and reinforcement learning (RL) respectively. We show that while the papers differ considerably in their approach, both demonstrate a substantial improvement over manual threshold-based autoscalers. In addition however, we point out reactiveness as a prohibitive weakness of the MLScale algorithm and identify safe and effective scaling as an area requiring further research to achieve industry-wide adoption of an RL-based solution.

In order to do so, the paper is structured as follows: We will explain requisite theoretical knowledge in Section 2 and discuss related works in Section 3. Section 4 introduces and contrasts the two algorithms in detail, closing with a conclusion and an outlook on future development in Section 5.

2. Background

In this section, the scientific concepts underlying this paper will be discussed. Specifically, we will focus on the current applications of cloud computing, as well as the theory behind two central paradigms of ML: supervised and reinforcement learning.

2.1. Cloud computing

The main advantage of cloud computing lies in the ability for the provider to easily and swiftly split computing resources, supplying a large amount of individual customers [1]. In order to simplify this process and allow division at a more granular level, CSPs utilize virtual machines and a process known as autoscaling.

2.1.1. Virtual machines. A virtual machine (VM) is a software-based emulation of the runtime environment provided by a physical computer. In contrast to a real computer however, the resources the VM has access to, such as the CPU cores or memory, can be varied by the underlying program. As a piece of software, multiple instances of VMs can be run on a single server, each of

doi: 10.2313/NET-NET-2022-07-1_11

which can be used independently and thus provided to a different customer [10, Chapter 1]. This virtualization is most common for servers, but has started to be increasingly applied to networking appliances, such as routers, switches or firewalls. Each instance of such virtualization applied to a networking service is called a virtual network function (VNF) [11].

2.1.2. Autoscaling. The virtual nature of the implementation allows for the easy up- and downscaling of the computing resources provided to a particular service as required by demand. This property is known as elasticity. Doing so automatically, or using automated policies, is called autoscaling. Applications can be scaled horizontally (in and out), representing the removing and adding of instances, as well as vertically (up and down), representing the addition or removal of an existing instance's resources [6, Chapter 8.2]. Autoscaling policies are geared towards a variety of different goals, such as improving resource utilization or decreasing operating expenses. Most notably, they aim to minimize service level agreement (SLA) violations, a type of contract stipulating the conditions of the service provided by the operator [12]. As Zhang et al. point out in [13, Section 6.1], in contrast to these complex targets, they often rely on rather simplistic metrics, such as throughput, response time, or amount of user requests, further increasing complexity.

2.2. Machine learning

Machine learning is a field of computer science focused on training an algorithm through the use of experiences and data, without programming explicit rules [14]. This approach is especially useful in environments where the ruleset is either particularly complex or not explicitly known by humans, such as natural language processing [15], computer vision [16], autonomous driving [17] or board games like Go [18], among a large variety of other use cases. There exist three basic paradigms of machine learning: supervised, unsupervised, and reinforcement learning. We will first focus on SL using neural networks and multiple linear regression, while the final section will introduce RL using SARSA and present an optimization technique called function approximation.

2.2.1. Supervised learning with neural networks. In contrast to the other two paradigms, in supervised Learning the algorithm is trained using data which includes the desired solutions [19, Chapter 1]. One such algorithm is called a neural network (NN). At a basic level, this algorithm tries to predict a set of m outputs (the solution), based upon a set of n inputs (the data). It consists of a set of simple processing units called neurons or nodes, which are organized into l layers. Each neuron ν can hold one value y. In a simple fully connected, feed-forward network each node $\nu_{i,j}$ of layer $i \in \{1, 2, ..., l-1\}$ is connected to every node $\nu_{i+1,k}$ of layer $i+1$, along with an individual adaptive weight $w_{i,j,k} \in [0, 1]$ associated with the connection from neuron j in layer i to neuron k in layer $i+1$. The first layer is called the input layer, as every node is initialized with one of the n inputs. Consequently, it consists of n neurons. Similarly, the last (l^{th}) layer is called the output layer, where each of the m neurons corresponds to one output. All layers in between are the hidden layers. Given all values $y_{i,1}, ..., y_{i,p}$ of layer i, the value of the k^{th} neuron in layer $i+1$ can be calculated using

$$y_{i+1,k} = f_{i+1}(w_{i,0,k} + \sum_{j=1}^{p} w_{i,j,k} y_{i,j}), \tag{1}$$

where f_{i+1} represents a non-linear function called the activation function and $w_{i,0,k}$ represents a weighted bias for layer i and neuron $\nu_{i+1,k}$. f is often varied on a per-layer basis. To train the NN, the individual connection weights $w_{i,j,k}$ and biases $w_{i,0,k}$ can be adjusted through a process called backpropagation until the prediction achieves good accuracy when compared to a target output (vector). To make a prediction, after initializing the input layer, we can consecutively calculate the values for successive layers, until the output layer is reached [19, Chapter 10] [20, Chapter 2].

2.2.2. Supervised learning with multiple linear regression. Prediction based on multiple linear regression (MLR) uses a set of data points in order to fit a linear function

$$\hat{Y} = a + \sum_{i=1}^{n} b_i X_i. \tag{2}$$

$X_1, ..., X_n$ represent a set of n inputs called explanatory variables along with associated weights $b_1, ..., b_n$. $\hat{Y}$ represents the predicted output called the response variable, and a is the bias [21].

2.2.3. Reinforcement learning. Reinforcement learning is a subcategory of machine learning, where an agent interacts with an environment [19, Chapter 1]. The states the world can be in, the actions that can be taken in it, as well as its time, are discretized into states $s \in S$, actions $a \in A$ and time steps t. Thus, at time time t, the agent can transition from states S_t to S_{t+1} using actions A_t. Accordingly, the process can be modelled as a markov decision process (MDP) [22] [23, Chapter 3.1].

The agent is further awarded a positive or negative reward R_{t+1}, based upon which it aims to construct an optimal action-value function

$$q_\star(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]. \tag{3}$$

$q_\star : S \times A \to \mathbb{R}$ reflects the expected total reward G_t, or in other words the value of an action given a certain state. Deriving from $q_\star$, the algorithm creates the optimal policy $\pi_\star : S \to A$, mapping every state to the action that will lead to the highest expected value. $\pi_\star$ is the final decision-making function [23, Chapter 4]. This approach to RL is called model-free, as it does not require an explicit model of the environment to learn, instead simply observing the rewards in relation to the given states and taken actions.

SARSA. The state-action-reward-state-action algorithm is one such model-free approach that aims to converge to $q_\star$. This estimate is denoted by Q. In order to do so, it starts with an initial function Q and updates the function every time an action is taken:

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{4}$$

doi: 10.2313/NET-NET-2022-07-1_11

It is proven that Q converges to $q_\star$ [23, Chapter 6.4] [24].

Function approximation. Whereas for smaller state spaces it can be enough to iteratively apply SARSA and update the policy function π, a process called policy iteration, this can become computationally infeasible even for comparatively simple tasks. The upper bound for greedy policy iteration is considered to be $O(\frac{k^n}{n})$ [25], where n is the number of states, and k the number of available actions per state. Instead of using a tabular representation of Q, function approximation utilizes a differentiable function

$$\hat{Q} : S \times A \times \mathbb{R}^d \to \mathbb{R} : (s, a, \mathbf{w}) \mapsto y, \tag{5}$$

where $\mathbf{w} := (w_1, w_2, ..., w_d)^T$ represents a vector of weights and y represents the calculated value. Applying stochastic gradient descent [26], $\mathbf{w}$ is updated after every action so that $\hat{Q}(s, a, \mathbf{w})$ approximates $Q(s, a)$ [23, Chapter 9]. However, as $d < |S|$, Q can often not be exactly approximated. This approach converges in $O(n^3)$ [27].

2.2.4. Parallel learning. In order to speed up convergence, N agents can interact with the environment at the same time and independently of each other [28]. In the implementation used by Benifa et al. [9], each agent j keeps track of its own local action-value function Q_{l_j}. Periodically, every agent shares its local estimate Q_{l_j} with every other agent, receiving the other local estimates $(Q_{g_1}, Q_{g_2}, ..., Q_{g_{N-1}})$ known as global estimates. To compute a final estimate Q_{f_j} every agent calculates a weighted average

$$Q_{f_j} = \frac{1}{2}(Q_{l_j} + \frac{\sum_{i=1}^{N-1} w_i Q_{g_i}}{N-1}). \tag{6}$$

This process is repeated until the final estimates for each agent converge to a single value [9, Section 3.1].

3. Related work

Given the wide applicability of a given solution, an extensive selection of related work is available. Singh et al. [29] and Qu et al. [30] provide a comprehensive analysis of autoscaling web applications in a cloud environment. Additionally, they introduce an expansive taxonomy further distinguishing between metrics, type, policy, and pricing, among other factors. Garì et al. [31] present an extensive survey of reinforcement learning-based autoscalers in particular, differentiating between model-free and model-based, sequential and parallel, as well as deep reinforcement and fuzzy logic learning. They conclude with a classification of the different approaches and provide a taxonomy based on their findings.

However, to the best of our knowledge, no survey exists which addresses and compares approaches in different machine learning paradigms specifically.

4. Comparison between MLScale and RLPAS

In the following chapter we will compare two contrasting approaches to autoscaling in cloud computing environments. The first paper, *"MLscale: A Machine Learning Based Application-Agnostic Autoscaler"* by Wajahat et al. [8], presents an algorithm that predicts key metrics, such as the response time, using simple application-independent inputs and makes autoscaling decisions based on the output. The prediction utilizes a simple neural network, combined with multiple linear regression for the decision making process. The second paper, *"RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment"* by Benifa et al. [9], defines a SARSA-based parallel reinforcement learning algorithm that tries to predict future workload, based upon which it scales the applications proactively.

4.1. MLScale: Autoscaling using Neural Networks and Regression

The algorithm introduced by Wajahat et al. consists of two different prediction algorithms: neural networks and multiple linear regression. In order to build the MLScale algorithm, Wajahat et al. split the program into 3 phases. Initially, the authors trained a neural network on a set of 8 application-independent input metrics $m_1, ..., m_8$, such as RR: number of requests received per second, or CPU: average CPU usage, in order to predict a single performance metric RT: the response time. The network consists of 8 input nodes, one 4-node hidden layer, and one output node. The activation function is $f(x) = \frac{1}{1+e^{-x}}$ and is only applied in the hidden layer.

To automatically scale any given application, MLScale continuously monitors all input metrics $m_1, ..., m_8$ and predicts RT. Should the response time exceed the target or even cause an SLA violation, MLScale will try to provision resources accordingly. To calculate the size of the additional resources, the program has to anticipate how this scaling will affect the new response time $\hat{RT}$. Thus, another prediction is necessary. Deriving from 2, the authors employ a simple multiple linear regression model predicting the new value $\hat{m}_i$ for every metric m_i after scaling:

$$\hat{m}_i = a + b_1 m_i \frac{w}{w+k} + b_2 m_i \frac{k}{w+k}. \tag{7}$$

$w \in \mathbb{N}$ represents the currently deployed workstations, while $k \in \mathbb{Z}$ represents the additional workstations. Positive k is to be understood as a scale-out and negative k as a scale-in. MLScale can not scale vertically.

Using the same network presented above, the predicted metrics $\hat{m}_1, ..., \hat{m}_8$ are then used to calculate the new $\hat{RT}$ after the scaling operation has concluded, based upon which the size of the scaling operation is decided.

4.2. RLPAS: Autoscaling using Parallel Reinforcement Learning

In contrast, Benifa et al. [9] attempt to construct an autoscaler using reinforcement learning. To define the MDP, the authors utilize a state set $S = \{U_{req}, U_{VM}, U_{RT}, U_{THR}\}$, representing the number of requests, percentage of allocated VMs, response time, and throughput. Additionally, the action set $A = \{A_{scale\_up}, A_{scale\_down}, A_{no\_change}\}$ is considered, each $A(VM_n, VM_{type})$ describing an amount VM_n and type $VM_{type} \in \{small, medium, large\}$ to be

doi: 10.2313/NET-NET-2022-07-1_11

scaled. VM_n represents horizontal scaling, while VM_{type} represents vertical scaling. The reward

$$R_t = \frac{Perf_{VM}}{U_{VM}} \tag{8}$$

is computed utilizing

$$Perf_{VM} = \frac{RT_{SLA}}{RT_{obs}} + \frac{THR_{obs}}{THR_{SLA}} - Penalty_{RT} - Penalty_{THR}. \tag{9}$$

The agent is thus rewarded for low RT and high throughput compared to the target. Meanwhile, the *Penalty* terms are applied when the agent exceeds the SLA thresholds. They are calculated based on the amount the SLA was violated by, as well as a manual weight. This ensures SLA-compliant behaviour, but also allows the operator to manually tune how severely a violation is to be punished, and thus, how close to the target the agent operates. 8 ensures that the agent does not overprovision VMs. As it covers all functionality laid out in Section 2.1.2, we believe this to be a sensible choice of reward function. In addition, 9 allows easy extension for other metrics.

The authors use a function approximation based on the gradient descent algorithm shown in Section 2.2.3, as well as parallel learning to considerably speed up convergence [9, Figure 9]. Because $q_\star$ is estimated by taking into consideration all future rewards as shown in 3, RLPAS can account for possible future developments of the input metrics. As such, this makes it a proactive algorithm, able to scale applications in preparation for incoming changes in request rate.

4.3. Discussion

Both approaches show very promising results for solving the problem of autoscaling according to user demand. Yet, in both solutions we were able to identify weaknesses which will require further research to address.

Utilizing easily obtainable input metrics, a small feed-forward NN, and MLP, MLScale [8] provides a prediction-based reactive autoscaling algorithm. In essence, Wajahat et al. present an ML-based approach to the traditional manual threshold-based autoscaling, allowing for an automated solution utilizing more metrics than would be possible by hand. For comparison, manual thresholds often rely on as little as one or two metrics [29, Section 5.2], compared to MLScale's 8. It's main advantage lies in this relative simplicity, as the architecture of the neural network can be trained in a few seconds. As the authors mention in [8, Section 3.1], the labeled training data can be acquired by sampling only a few hours worth of normal application behaviour, ideally including a wide variety of workload and scaling actions.

However, this simplicity also leads to its largest deficiencies. For one, the paper only considers a single target metric: response time. Yet, in order to achieve the complex goals set for the algorithm, other performance indicators such as CPU utilization, throughput, or power consumption should also be taken into consideration. While the authors argue in [8, Section 3.1] that the NN could be easily extended to account for these metrics, we expect this to increase both complexity and training time. In addition, the presented method represents a reactive autoscaler, meaning the algorithm does not predict future workloads and can only react to them once they occur. As shown by Wajahat et al. in [8, Section 5], MLScale incurs a substantial (up to 5.9%) amount of SLA violations especially during large and sudden request spikes, also tending to overprovision resources in response. While this still represents an improvement in comparison to traditional scalers and achieves near optimal performance for workloads less prone to spiking [8, Table 5], the problem remains near impossible to solve using reactive methods.

In contrast, the algorithm presented by Benifa et al. [9] is of a proactive nature. As the authors are able to show in [9, Section 4.4] and especially [9, Figure 7], after an initial phase of fluctuation, RLPAS is able to achieve very steady performance metrics even for rapidly changing workloads. Unfortunately the authors do not quantify the number of SLA violations, which would allow a more direct comparision to MLScale. However, as the measurements suggest highly stable target metrics, we have no reason to believe any substantial amount of violations occurred. In addition, this approach is able to outperform competitors in a variety of different measurements, including RT, throughput, and CPU utilization. Importantly, RLPAS can scale both horizontally and vertically, while MLScale is restricted to horizontal scaling. This increases the flexibility of the algorithm and enhances its ability to find a good solution.

However, in comparison with the simple mechanisms used by MLScale, RLPAS is a more algorithmically complex solution. As Sutton points out in [23, Part II], major weaknesses of reinforcement learning tend to be two-fold: (1) poor initial performance as the agent begins to explore the environment and the Q-table values have not yet converged and (2) long convergence time. Especially for very large state spaces, the consequently larger value table can result in very long training times, thus further amplifying the first weakness (see Section 2.2.3). The authors attempt to combat this problem by combining function approximation and parallel learning in order to speed up convergence. Nonetheless, the paper only demonstrates acceptable convergence for up to 16 VMs per application [9, Figure 8] as well as a limited set of 4 state and 3 action variables [9, Section 3.1]. As this convergence issue is a well-known problem faced by the wider RL community, further research is required to show acceptable performance in such situations.

5. Conclusion

In conclusion, both machine learning-based solutions in general, and the analyzed algorithms specifically, provide a marked improvement over manual, threshold-based autoscaling methods. However, as we have shown in the preceding sections, the reactive nature of the MLScale algorithm represents a major weakness preventing its adoption in favor of more advanced, proactive algorithms, such as RLPAS. In addition, RLPAS' ability to predict multiple different important metrics, as well as scale both horizontally and vertically, gives it a further edge when optimizing for complex scenarios. As such, we conclude from our analysis that exploration in the direction of reinforcement learning appears the most promising. Although having already produced encouraging results, we believe further research into the safe and effective scaling is required to achieve industry-wide adoption.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011-09-28 2011.

[2] M. J. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models*, 1st ed., ser. SaaS, PaaS, and IaaS. Wiley, 2014.

[3] Amazon, "Aws pricing," accessed: 2021-12-02. [Online]. Available: https://aws.amazon.com/pricing/

[4] "Current enterprise public cloud adoption worldwide from 2017 to 2020, by service," Flexera Software, Mar. 2021, accessed: 2021-12-02. [Online]. Available: https://www.statista.com/statistics/511508/worldwide-survey-public-coud-services-running-applications-enterprises/

[5] Statista, "Cloud computing market size in europe from 2016 to 2025, by segment," Aug. 2021, accessed: 2021-12-02. [Online]. Available: https://www.statista.com/forecasts/1235161/europe-cloud-computing-market-size-by-segment

[6] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st ed. Addison-Wesley Professional, 2015.

[7] Oracle, "Oracle autoscaling," 2022, accessed: 2022-02-25. [Online]. Available: https://docs.oracle.com/en-us/iaas/Content/Compute/Tasks/autoscalinginstancepools.htm

[8] M. Wajahat, A. Karve, A. Kochut, and A. Gandhi, "Mlscale: A machine learning based application-agnostic autoscaler," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 287–299, 2019. [Online]. Available: https://doi.org/10.1016/j.suscom.2017.10.003

[9] B. Benifa, J. V., and D. Dejey, "Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, vol. 24, pp. 1348–1363, 2019. [Online]. Available: https://doi.org/10.1007/s11036-018-0996-0

[10] R. N. Jim Smith, *Virtual Machines - Versatile Platforms for Systems and Processes*, 1st ed., ser. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2005.

[11] K. Gray and T. D. Nadeau, *Network Function Virtualization*. Elsevier Science & Technology; Morgan Kaufmann, 2017.

[12] W. T. Joe M. Butler, Ramin Yahyapour, *Service Level Agreements for Cloud Computing*, 1st ed. Springer-Verlag New York, 2011.

[13] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, Apr. 2010.

[14] T. M. Mitchell, *Machine Learning*, 1st ed., ser. McGraw-Hill series in computer science. McGraw-Hill, 1997.

[15] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021.

[16] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3d data: A survey," *ACM Comput. Surv.*, vol. 50, no. 2, apr 2017. [Online]. Available: https://doi.org/10.1145/3042064

[17] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, Apr. 2020.

[18] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[19] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O'Reilly Media, 2017.

[20] K. Gurney, *An Introduction to Neural Networks*. UCL Press, 1997.

[21] G. Seber and A. Lee, *Linear Regression Analysis*, 2nd ed., ser. Wiley Series in Probability and Statistics. Wiley, 2003.

[22] R. Bellman, "A markovian decision process," *Indiana University Mathematics Journal*, vol. 6, pp. 679–684, 1957.

[23] R. S. Sutton, *Reinforcement Learning: An Introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[24] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 287–308, 2000.

[25] Y. Mansour and S. Singh, "On the complexity of policy iteration," *UAI*, 01 2013.

[26] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning*. MIT Press, 2016.

[27] A. Haider, G. Hawe, H. Wang, and B. Scotney, "Gaussian based non-linear function approximation for reinforcement learning," *SN Computer Science*, vol. 2, no. 3, pp. 1–12, 2021.

[28] R. M. Kretchmar, "Parallel reinforcement learning," in *The 6th World Conference on Systemics, Cybernetics, and Informatics*. Citeseer, 2002.

[29] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, "Research on auto-scaling of web applications in cloud: Survey, trends and future directions," *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, 2019.

[30] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018.

[31] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. G. Garino, "Reinforcement learning-based application autoscaling in the cloud: A survey," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104288, 2021.

Ultra-Low Latency on Ethernet Technology

Atilla Alpay Nalcaci, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: atilla.nalcaci@tum.de, wiedner@net.in.tum.de

***Abstract*—Network latency depicts the total amount of time for a data packet to be captured, processed and transmitted, potentially through multiple devices, from one communication endpoint to another. This measurement of delay is a performance characteristic among telecommunications and cellular communication providers.**

In this paper, we present our research on the implementation requirements of Ultra-Reliable Low-Latency Communication (URLLC) to the current ethernet infrastructure. Further, we analyze commodity software and hardware on the performance of low latency packet processing. Investigations focus on network areas and quality of service provisions and conclude on requisites to support URLLC applications in shared networks. Findings show that any non-specialized network infrastructure requires fine-tuning of communication specifications that is capable of achieving maximum transmission delay of approximately 50 ms with very high achievable network reliability and utilization measurement.

***Index Terms*—5G, ultra-reliable low-latency communication, network latency, packet processing, reliability**

1. Introduction

The latency of a network describes the overall delay in the communication, usually measured in ms (millisecond) and the final result is typically indicated as a round trip delay – the absolute amount of time that is spent for transmitting the information to the target destination and then back to the original sender. It is important to appertain performance optimizations concerning the latency to test system performance emulating under high latency in order to optimize for users with lousy connections.

Ultra-low latency is a service category introduced in 5G New Radio (NR) standard which allows newly emerging services and applications to surpass and resolve the prospective latency and reliability requirements. 5G NR is the global standard for a robust and capable cellular network infrastructure that enables enhanced communication between user endpoints in terms of data delivery, reliability, and transcend user experience on a massive scale [1]. In summary, 5G networks encapsulate the following generic connectivity types: enhanced Mobile Broadband (eMBB), massive Machine-type Communication (mMTC), and Ultra-Reliable Low-Latency Communication (URLLC) [2].

The conception of 5G networks is inclined to interweave with the notion of "ultra-reliable" connectivity, making the implementation process of URLLC rather difficult and restrictive [3]. The trend of ultra-reliable communication guarantees perpetual connectivity of approximately > 99.999% for a given time window [4]. URLLC enables computer networks to process and exchange high volume data packets with eminently low latency between the endpoints. These networks support real-time access and request/response to previse rapidly changing data [2].

The key feature of URLLC is low latency. This is a crucial aspect for devices and/or gadgets which perform over a common network of command nodes that provide query of commands on what needs to be executed next [2]. Performance measurements that are included in the following sections are conducted in the context of tail latency-percentage of response times out of all responses to the input and output requests that the system serves, which take the longest amount of time in comparison with the totality of its response times. With low tail latency, networks are open to optimizations that enables the processing of large amounts of data with minimal latency. Since networks are required to be adaptive to dynamic data entries and alterations, these optimizations have the potential to increase the overall network utilization as well as inaugurate an expeditious method of data transfer.

In this paper, we present our research and analysis for the requirements of URLLC to the current ethernet technology. Further, we analyze what is needed to support URLLC applications in shared networks. The paper is organized as follows. Section 2 represents some background and related work. Section 3 examines the current status and evolution of the ethernet technology. Section 4 gives a brief description and potential sources of tail latency. Section 5 presents thorough information on the prerequisites of URLLC network infrastructure. Finally, Section 6 concludes the paper and provides some literature on various future work.

2. Background and Related Work

In this section, we present the work on supporting URLLC on non-specialized networks and latency measurement methodologies for low-latency systems.

2.1. Latency Measurement

Several studies exist [5]–[7] for achieving highly reliable connectivity with low-latency measurements. The research carried out by Gallenmuller et al. [8] depicts a new methodology for measuring the tail latency of Linux

doi: 10.2313/NET-NET-2022-07-1_12

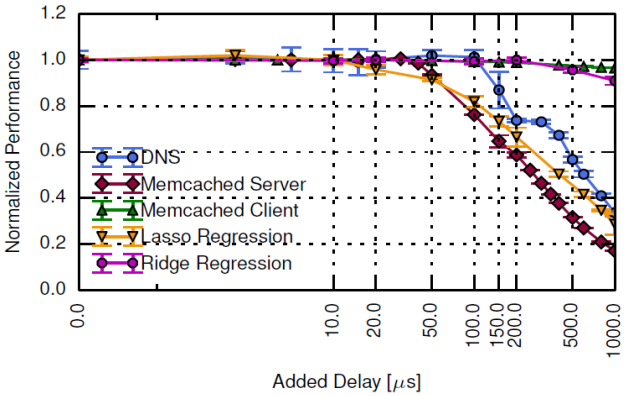


Figure 1: The effect of static latency on different applications. [9]

supported off-the-shelf hardware commodities. Furthermore, the research presents a software stack that lowers the overall tail latency of packet processing applications. Latency measurements are made through hardware timestamping for increased precision. The software stack that is presented as a solution has attested to the occurrences of low-latency packet processing on a consistent demeanor. Ensuing case study proved to achieve a forwarding latency of below 25 µs for a non-overloaded Snort IPS.

2.2. Operating System and Hardware

Identifying the software- and hardware-related latency and jitter is one challenge of ensuring low latency while keeping the connectivity uninterrupted, i.e. reliability of the connection. As Stylianopoulos et al. [4] examine, the main objective is to prevent the network interruptions that are directly influential over the user-space applications which are responsible for handling the packet processing flow in its service, to the furthest extent possible. To achieve this, certain kernel options are introduced and later delineated to have contributions to lower and stable network latency. Examples include preparatory configurations of system-level setup options that are namely; Thread isolation which isolates the Data Plane Development Kit (DPDK) cores to prevent common use of these cores by other tasks, disabling of interrupt balancing for disabling the dynamic interrupt distribution daemon to avoid unrelated DPDK interrupts, and disabling Intel turbo-boost technology which introduces high variation to packet processing latency.

2.3. Cloud-based Applications

Cloud-based applications are described as the software which the analogous users access through a shared network, commonly being the internet. The research carried out by Popescu et al. [9] focuses on characterizing the latency of the cloud-based applications' performance. Applications that are used during this research are Domain Name System (DNS), Memcached, STRADS-a scheduled model parallelism distribution framework, and Apache Spark. The methodology is based on devising the host to experience different network latency values by modifying the link that connects the Top of Rack switch (ToR) to the corresponding host. Figure 1 gives an overview of the mentioned applications that are experimented in terms of their additive latency – x-axis is the static latency added in microseconds for round-trip time (RTT) and y-axis is the normalized performance. In particular, the baseline performance of the individual applications is analogized with the ratio of the measured performance at each latency point to analyse the effect of static latency [9].

Experimental conclusions suggest that different injections of controlled network latency have varying impacts on different applications. In particular, latency values are affected to differing amounts, such that even small network delays are found to be influential upon divergent application performance, nearly tens of microseconds.

TABLE 1: Throughput and latency in 1G to 5G [11]

Generation	Data Rate/Throughput (Maximum)	Latency (Minimum)
1G	9.6 kbit s^{-1}	$>$ 1000 ms
2G	2 Mbit s^{-1}	600−750 ms
3G	100−300 Mbit s^{-1} (DL), 50−75 Mbit s^{-1} (UL)	$<$ 10 ms (UP), $<$ 100 ms (CP) (typical values: 40−50 ms)
4G	1−3 Gbit s^{-1} (DL), 0.5−1.5 Gbit s^{-1} (UL)	$\sim$ 5 ms (UP), $<$ 100 ms (CP) (typical values: 40−50 ms)
5G	1 Tbit s^{-1} (over 100 m) $>$ 20 Gbit s^{-1} (DL), $>$ 10 Gbit s^{-1} (UL)	$\leq$ 1 ms

3. Current Status of Ethernet Technology

Presently, ethernet is the most widely used commodity network system that allows the implementation of wired computer networking technologies, most of which are commonly being used in local area networks (LANs) and wide area networks (WANs). Capabilities of the modern ethernet technology allow expeditious data transfer and hard real-time communication. Ethernet is readily scalable, thereby enabling thriving technologies to be easily integrated. Subsequently, as Loeser and Haertig [10] points out, the current ethernet infrastructure is increasingly moving towards the switches–network connection devices that manage the data flow in a given network by transmitting data packets between corresponding hosts. In the context of media-access control, modern ethernet technology uses Carrier-sense multiple access with collision detection (CSMA/CD) to defer data transmissions until the predefined communication channel is not occupied by any transmission. The aforementioned shift to network switches allow the use of traffic shaping strategies by means of implementing the hard real-time distributed systems on commodity networks. Nevertheless, current intuition regarding the collision avoidance limitations yields an increase in terms of processing load and bandwidth allocation over a common network.

The evolution of network systems and their specifications has been comprehensive apropos the changing network architectures and radio access network (RAN)

systems [12]. Throughout different generations of network evolution, two principal parameters exist that are rudimentary, namely throughput and latency. While latency signifies the amount of time for data to travel from one communication endpoint to another, throughput denotes the amount of data that has moved successfully between the predetermined hosts. With the drastic evolution of communication technologies, significant architectural changes eventuated, introducing seamless connectivity and mobility properties. In summary, Table 1 shows the values of different generations concerning the conjectural throughput and latency evaluations [11].

Latest generation systems are intended to achieve efficient system development and utilization, in addition to preserving end-to-end connection requirements. Nonetheless, the scope of this research does not cover deployment and optimization fields, as the main focus is the application of URLLC on non-specialized networks.

4. Sources of Tail Latency

The presented rationale is gathered from multiple tuning guides, while also remarking the presence of various studies that aim to overcome ambiguous extents that are arisen from the complication of tail latency.

As already outlined, tail latency, commonly referred to as high-percentile latency, is the percentage of response times from which the response is received that takes the longest amount of time in contrast to the overall response times of the specified server. Maintaining a low margin for tail latency is tricky, especially for large-scale applications that consist of interactive operations. Tail latency is considered to be problematic due to numerous reasons. As outlined by Haque et al. [13], applications with interactive foundations contend in terms of providing complex user functionalities under strict latency constraints. As a result, this creates an unavoidable setting in which tail latency having an impact over user requests pursuant to high degrees of parallelism—a performance metric that indicates the number of operations that can be executed on a server concurrently [13]. Since the totality of a request is not finalized until the slowest sub-request is finished, tail latency is proved to be an arduous challenge for developers.

While tail latency might be an outcome of an application-specific service, there are numerous reasons where tail latency can be introduced to a network, some examples being hardware peripherals, operating system kernel modes or application-level configuration preferences. For instance, as Li et al. [14] points out, buffering has an immense impact on networks that have low traffic rates. This is considered as a primary predicament since URLLC applications generally possess low traffic rates, interpreting an operose situation since such conditions is critical.

Nonetheless, there are numerous studies [10], [13], [15] that explore and mitigate the problem of high latency through modifying certain kernel operations, using various software development tools to enable ISP and P2P user cooperation and implementation of traffic shaping on switched Ethernet.

5. Network Communication Requirements of URLLC

Following explorations and analysis are gathered from various articles that focus on network latency characterization and URLLC performance on commodity hardware. Imperative network specifications and requirements are listed, and use cases are denoted accordingly.

The main purpose of URLLC is to resolve newly emerging latency-critical applications by means of handling the prospective latency and reliability requirements. In principle, network systems that support URLLC applications are capable of supporting real-time access to rapidly changing data by design, thereby allowing the network to be optimal and available to network optimizations in the context of processing high volume data packets with eminently low latency [4]. While the benefits of URLLC on a network are authenticated, particular communication requirements must be established to the network before enabling URLLC supported applications. Preliminary requirements of URLLC services that are prospective to the network infrastructure which the URLLC will be deployed are as follows:

a Low latency: The approximated maximum end-to-end latency requirements for a network with URLLC adaptation, ranging from 1 ms to 50 ms. On average, the conception of URLLC requisites presented by 3rd Generation Partnership Project (3GPP) organizations is an average user-plane radio latency of 0.5 ms, comprising uplink and downlink together. Note that these values are not bounded by an associated reliability value [4].

b High reliability: As stated in section 1, trend of URLLC guarantees a perpetual connectivity ratio of nearly $> 99.999\%$ reliability. As reported by Stylianopoulos et al. [4], URLLC use cases stipulate a reliability measure, ranging from 99.9% to 99.999% of reliability. Note that the depictions are based on a network latency of 1 ms for a transmission of packet size 32 bytes. Thence, the network infrastructure must be capable of sustaining a highly reliable packet delivery margin.

c Low jitter: In spite of the general network specifications, i.e. a network infrastructure which is verified to maintain a latency extremity that is in the acceptable bounds of a system, a certain deviation from the true periodicity of a network is prospectively contingent [4]. This deviation is commonly referred to as "jitter" which describes the variance in latency. High values of jitter connote inadequate network performance and introduces packet loss to the network flow. In particular, communications services that are based on URLLC service category require the average jitter to be $< 50\%$ cycle time [16].

Additionally, low traffic rates are also another aspect that is considered essential to the notion of URLLC services. However, event-based applications which augmented to function in a dynamic environment are not

TABLE 2: Example of low latency and high reliability use cases and their requirements [1]

Scenario	End-to-end latency	Reliability
Discrete automation— motion control	1 ms	99,9999%
Electricity distribution— high voltage	5 ms	99,9999%
Remote control	5 ms	99,999%
Discrete automation	10 ms	99,99%
Intelligent transport systems— infrastructure backhaul	10 ms	99,9999%
Process automation— remote control	50 ms	99,9999%
Process automation— monitoring	50 ms	99,9%
Electricity distribution— medium voltage	25 ms	99,9%

both latency and throughput critical. Standard use cases of these applications foster an approximate broadband speed of $< 50\,\mathrm{Mbit\,s^{-1}}$, a comparatively low traffic rate as contrasted with modern networks [1].

An example of URLLC use cases and requirements [1] are depicted in Table 2. Examples are made with respect to predefined industrial applications that benefit from the utilization of a URLLC network infrastructure. As formerly indicated, end-to-end latency values are in the range of 1 ms to 50 ms, in addition to the eminent reliability percentages. Per contra, maintaining the scope on network design and overall system performance.

6. Conclusion and Future Work

Ultra-reliable and low-latency communication is a substantial service category for providing reliable connection segments to applications that retain stringent latency and reliability measures. The main features of URLLC, low latency and high reliability in particular, enables a primary usage scenario for 5G network infrastructure. In order to sustain a network system that supports URLLC applications, the analogous network infrastructure must be fine-tuned in terms of sustaining high reliability for the correspondent application channels and a latency measurement of 50 ms extremity. Furthermore, certain studies exist for enabling and testing URLLC on Wireless Access systems and Cloud-based application data centers.

In particular, Popovski [17] provides a framework that can be utilized for scheming ultra-reliable wireless network systems, and analyzing accordingly. Previous work of the same research depicts the building blocks for the appliance of URLLC in wireless network access. Continually, the following research is aimed to provide further information on techniques and principles for URLLC wireless access. The research further annexes investigations by introducing a detailed discussion on communication-theoretic principles of URLLC. Subsequently, concepts of latency and reliability are expressed as coupled, from the perspective of an application that has a predefined latency constraint. At length, reliability of a communication is defined under the probability that the measured latency does not exceed this predefined latency constraint.

Additionally, Popescu et al. [9] present quantitative results regarding test benchmarks of cloud-based applications. Results suggest that applications that are of different complexity and distance to the corresponding data centers are affected by network latency to differing amounts. Findings are auspicious with respect to sustaining a cloud-based ultra-low latency network environment for the upcoming future studies that are fundamental to this area.

References

[1] Z. Li, H. Shariatmadari, B. Singh, and M. A. Uusitalo, "5G URLLC: Design Challenges and System Concepts," pp. 1–5, 2018, accessed: 2021-12-08. [Online]. Available: http://dx.doi.org/10.1109/ISWCS.2018.8491078

[2] P. Popovski, C. Stefanovic, J. J. Nielsen, E. de Carvalho, M. Angjelichinoski, K. F. Trillingsgaard, and A. S. Bana, "Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC)," vol. 67, no. 8, pp. 5783–5786, 2019, accessed: 2021-11-18. [Online]. Available: http://dx.doi.org/10.1109/TCOMM.2019.2914652

[3] P. Popovski, J. J. Nielsen, C. Stefanovic, E. de Carvalho, E. Strom, K. F. Trillingsgaard, A. S. Bana, R. Kim, D. M. Kotaba, J. Park, and R. B. Sorensen, "Ultra-Reliable Low-Latency Communication (URLLC): Principles and Building Blocks," pp. 1–7, 2017, accessed: 2021-12-05. [Online]. Available: http://dx.doi.org/10.1109/MNET.2018.1700258

[4] C. Stylianopoulos, M. Almgren, O. Landsiedel, M. Papatriantafilou, T. Neish, L. Gillander, B. Johansson, and S. Bonnier, "Industry Paper: On the Performance of Commodity Hardware for Low Latency and Low Jitter Packet Processing," pp. 1–5, 2020, accessed: 2021-11-18. [Online]. Available: http://doi.org/10.1145/3401025.3403591

[5] AMD, "Performance Tuning Guidelines for Low Latency Response on AMD EPYC 7001-Based Servers - Application Note," 2018, accessed: 2021-11-12. [Online]. Available: http://developer.amd.com/wpcontent/resources/56263-Performance-Tuning-Guidelines-PUB.pdf

[6] J. Mario and J. Eder, "Low Latency Performance Tuning for Red Hat Enterprise Linux 7," 2017, accessed: 2021-11-12. [Online]. Available: https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v2.1.pdf

[7] E. Rigtorp, "Low latency tuning guide," 2021, accessed: 2021-11-12. [Online]. Available: http://rigtorp.se/low-latency-guide/

[8] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems," pp. 1–7, Oct. 29, 2021, accessed: 2021-11-12. [Online]. Available: http://dx.doi.org/10.23919/CNSM52442.2021.9615532

[9] D. A. Popescu, N. Zilberman, and A. W. Moore, "Characterizing the impact of network latency on cloud-based applications' performance," vol. 2, no. 914, pp. 3–16, Nov. 2017, accessed: 2021-11-12. [Online]. Available: http://dx.doi.org/10.17863/CAM.17588

[10] J. Loeser and H. Haertig, "Low-latency Hard Real-Time Communication over Switched Ethernet," pp. 1–3, 2004, accessed: 2021-11-12. [Online]. Available: http://dx.doi.org/10.1109/EMRTS.2004.1310992

[11] A. Slalmi, H. Chaibi, A. Chehri, R. Saadane, G. Jeon, and N. Hakem, "On the Ultra-Reliable and Low-Latency Communications for Tactile Internet in 5G Era," vol. 176, pp. 3853–3862, 2020, accessed: 2021-12-08. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050920318925

[12] O. T. Eluwole, N. Udoh, M. Ojo, C. Okoro, and A. J. Akinyoade, "From 1G to 5G, What Next?" vol. 45, Aug. 2018, accessed: 2021-12-10. [Online]. Available: http://www.iaeng.org/IJCS/issues_v45/issue_3/IJCS_45_3_06.pdf

[13] M. E. Haque, S. Elnikety, Y. h. Eom, R. Bianchini, Y. He, and K. S. McKinley, "Few-to-Many: Incremental Parallelism for Reducing Tail Latency in Interactive Services," pp. 1–4, 2015, accessed: 2021-12-05. [Online]. Available: http://dx.doi.org/10.1145/2694344.2694384

[14] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency," pp. 1–10, 2015, accessed: 2021-12-05. [Online]. Available: http://dx.doi.org/10.1145/2670979.2670988

[15] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P Users Cooperate for Improved Performance?" vol. 37, no. 3, pp. 31–34, 2007, accessed: 2021-11-12. [Online]. Available: http://dx.doi.org/10.1145/1273445.1273449

[16] L. Xia, X. Hou, G. Li, Q. Li, L. Sun, W. Rui, J. Erfanian, S. Tatesh, B. Liu, A. Chan, B. Tossou, A. G. Serrano, B. Sayrac, G. Wannemacher, A. Kadelka, A. Frisch, J. Sachs, D. Patel, and R. Sabella, "5G E2E Technology to Support Verticals URLLC Requirements," Nov. 18, 2019, accessed: 2021-12-14.

[17] P. Popovski, "Ultra-Reliable Communication in 5G Wireless Systems," pp. 1–4, 2014, accessed: 2021-11-20. [Online]. Available: http://dx.doi.org/10.4108/icst.5gu.2014.258154

Current State of Network Support in WebAssembly

Elias Nechwatal, Kilian Holzinger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: nechwata@in.tum.de, holzinger@net.in.tum.de

***Abstract*—WebAssembly is a binary code that is system independent. Many programming languages can be converted into WebAssembly (Wasm) and run on a browser or in a runtime outside the browser. However, some programs need networking support. Compiled into a Wasm module, these needs have to be mapped to the according resources that are accessible from the specific underlying system. This paper mainly focuses on this mapping with the goal to understand how networking functions can be reached. For a browser environment this would be the Web APIs, an interface provided by the browser, that grants access to system resources. Outside the browser there is an API called WASI that defines methods for system access that need to be implemented by the runtime. Currently, networking directly from WebAssembly fails on multiple aspects:**
- Web APIs: Wasm can not access the Web APIs currently
- WASI: WASI can be accessed, however, there is currently only few networking functionality defined
- Runtimes: If WASI defined enough networking functionality, runtimes still needed to implement it which is currently not the case.
A few options to network in Wasm, despite of these problems, is either to do it in JavaScript in a browser or to include a library that allows delegating http requests to a different program for experimental use outside the browser.

1. Introduction

In the past, several attempts were made to compile arbitrary code to JavaScript (JS). This seemed necessary because JS is an extremely popular programming language that is natively supported on the web [1]. The toolchain Emscripten [2] made it possible to compile some programming languages, such as C and C++, to asm.js, a JavaScript subset. However, asm.js reached its limits in terms of performance improvement, portability and security. That is because major changes and adding new features often would result in also changing JavaScript [1]. That is why asm.js was left behind and WebAssembly was invented. WebAssembly is an advanced portable binary format that several programming languages can compile to. Also known as Wasm, it surpassed the old project asm.js in all design goals. That is because Wasm has its own binary and can add features without having to depend on other languages unlike asm.js [1].

WebAssembly wants to make it possible, that code for many different purposes can be run in Wasm runtimes. However, programs that are compiled into Wasm could need all kinds of interfaces towards the system resources. In this paper we specifically focus on the possible ways to access network support from Wasm. Therefore we, on the one hand, look at the browser environment and its interfaces. On the other hand, from Section 5 on, we address Wasm outside the browser and how it is currently possible to network from there.

2. WebAssembly Design

For the understanding of how Wasm networking works, the fundamentals of WebAssembly are important. In that respect, a short summary of important aspects of wasm follows.

2.1. Binary Encoding

"The binary encoding is a dense representation of module information that enables small files, fast decoding, and reduced memory usage" [3]. Although Wasm is a binary encoding, Wasm can be also represented in a human readable text format called '.wat' [4]. Given a .wat file, binary encoding generates a .wasm binary file through formal grammar translation mechanisms. Wat modules have a tree like structure, as "a module is represented as one big S-expression" [4]. The nodes and the whole tree is formed by parentheses. Every command has its opcode and they are often similarly named as for instance the x86 assembly language. Nevertheless, coding in the text based .wat format is more comfortable, due to the simplified function declarations and import/export of modules.

2.2. Modules

A module consists of multiple sections, for instance an import section, to declare imported modules, or a function section, where all function signatures are declared. Writing text based WebAssembly by hand is rather unusual. However, when writing own modules and functions, one needs to be aware of Wasm code being designed to run on a stack based virtual machine. That means, a function pops its arguments from the operand stack and pushes the return value. It is shown that using a stack machine, files are smaller and decoding gets simpler than using a register machine [1] [5].

In WebAssembly, modules are used for any possible functionality. APIs and self written functionality are all included as modules.

doi: 10.2313/NET-NET-2022-07-1_13

2.3. Portability and Runtimes

WebAssembly is portable, as it "does not specify any APIs or syscalls" [6]. However, every access to the system resources that a module requests, has to go through APIs. Without these, the module would not be able to run.

For instance, assume a C code converted into Wasm using memory allocation on the heap. This code could not run in a web environment without a WebStorage API or some sort of access to memory. WebAssembly lets the host choose the APIs, so that he can restrict usage of resources. Modules that were compiled to Wasm have to rely on the compiler, to map their interface calls to the available APIs [6].

Networking would also come as an API, the host runtime needs to offer. How exactly the implementation of an API looks like does differ from runtime to runtime and also for each opeating system. That is caused by the portable design of Wasm. It is not enough to compile the .wat textfile into different binary files for different machines to interact with (That is the way C code is compiled). That approach would prevent the portability of the generated .wasm file. To be able to run the Wasm binary code on any operating system, the runtime has to differ for each system. In that way, the runtime can delegate the general Wasm system calls to the special underlying system. The neccessary abstraction for the Wasm binary code is provided [7].

2.4. Security

Even though being associated with assembly languages, WebAssembly is considered as secure [8]. Not being able to run on a system directly, Wasm modules can only interact with the environment over well defined APIs. All modules run in their own sandbox and can only communicate with each other via imports. Unlike in C, Control-Flow-Integrity is always checked during runtime. To achieve this, a table, built during compilation, lists all functions that could possibly be called in the program which are compared with the functions that actually get called [8]. There are also measures against other possible attacks which will not be discussed further.

2.5. Use Cases

In general, WebAssembly is used whenever it comes to time critical calculations inside the browser [9]. Security also plays a role for developers that decide to use WebAssembly for implementations. Currently, many examples of existing code bases that are brought to the web can be found [10].

The usual way to implement Wasm is to code the website in JavaScript and outsource computation intensive tasks into Wasm, using the Web Embedding API and the JavaScript API of WebAssembly. In non-web environments, Wasm is often used as a secure way to mediate between a program and the host system as Wasm delivers another abstraction from the real system [11]. (Further explanation in Section 5)

3. Browser Environment

In the browser, WebAssembly needs to reach the host systems resources in order to perform networking. This is done with APIs explained in the following.

3.1. APIs for Web Embedding

In Figure 1, it is modeled how WebAssembly (in the browser) interacts with various other components of a website. Every arrow is an API one has to use, to access the other language/format. WebAssembly, for the browser, is just another document to receive from the webserver amongst the usual JS, CSS or html files. JavaScript can access the html file. This programming interface to the html document is called DOM (Document Object Model). Furthermore JS has an API to WebAssembly (JS API/WebEmbedding API), which is used in Websites that stick to JS, but want to have fast calculations in WebAssembly. However, looking at the WebAssembly's APIs there is no such arrow to the DOM. (This problem is further explained in Section 3.3). The workaround that allows to still access the DOM via Wasm is straightforward. By using the JavaScript Interface to access the DOM, the problem is solved.

As Wasm runs on the browser, there are many interfaces towards the resources that the host system offers. These are called Web APIs and are offered to websites and applications by the browser. Providing indirect access to system functions, such as memory allocation, networking and much more, they are essential for WebAssembly code [13].

3.2. Web Interface Description Language Bindings

However, accessing Web APIs, WebAssembly can not get along without JavaScript. In reality every Web interface access of WebAssembly gets executed by Javascript and not directly by Wasm. The reason for this problem are the limited types that WebAssembly provides. On the one side is an API coded in all possible programming languages and on the other side Wasm with only a very limited amount of data types (only integer and float). Mediating is difficult here. To deal with the different types of Web APIs, there is a standardization for structure of those types, called Web Interface Description Language. For JavaScript, there is a mapping from the JS types to the Web IDL types. Every time a function of the Web API is called, the JS parameter gets translated to a Web IDL type. The return type of the function also has to be converted in this manner. [14] That is where the problem for Wasm lies. "Currently, there is no mapping between WebAssembly types and Web IDL types. This means that, even for simple types like numbers, your call has to go through JavaScript." [14]. The translation of the types is also called bindings. In case of Javascript this is the ECMAScript Binding. It is obvious, that this process is time intensive. It takes time to get the parameters from WebAssembly, convert them to JavaScript types with JS gluecode and convert these types to the Web IDL format with the ECMASript Binding. This process also has to be

doi: 10.2313/NET-NET-2022-07-1_13

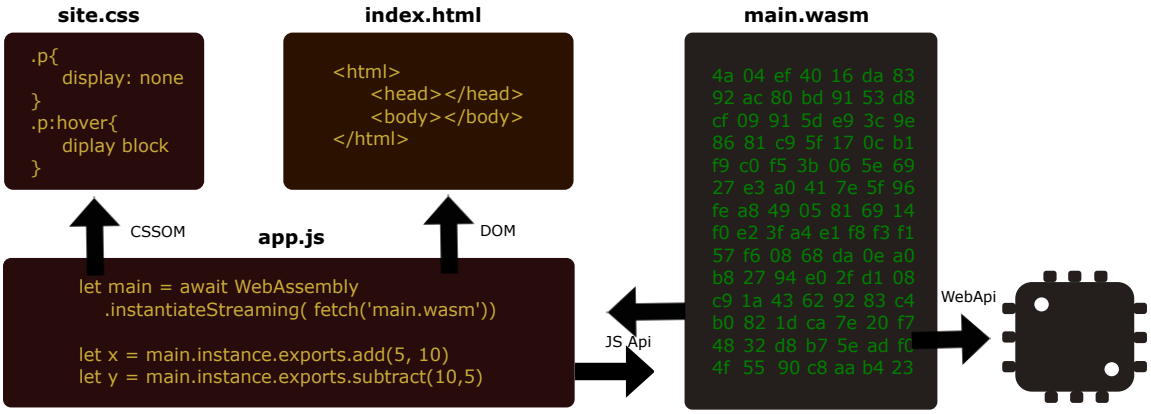


Figure 1: APIs in a browser [12]

reverted for the return value [15]. Another aspect is the memory usage, as the parameters have to be copied from the Wasm memory to JS heap to the renderer's heap.

Currently, this is the only way to access these Web APIs from Wasm. However, web developers are aware of this problem and try to create a mapping directly from the Wasm types to the IDL types. Nevertheless, this problem will not be solved soon as there are many difficulties mapping the types. "To have a straightforward mapping between WebAssembly types and Web IDL types, we'd need to add some higher-level types. And we are doing that — with the GC proposal" [14], another Problem explained next.

3.3. Garbage Collection in WebAssembly

WebAssembly has no Garbage Collection (GC) implemented nor any memory management tools. That is no problem for compiling languages into WebAssembly, that do not have GC either and clean their memory usage in the code. However, languages that are used to more support than just a linear piece of memory are struggling to compile to Wasm. "Currently, languages that require a GC have no other option than to compile the GC to .wasm and ship that as part of the binary" [16] which is not very efficient and increases the legth of the .wasm binary file. "However, the option of integrating with the host's GC will not only make it easier to compile numerous high-level languages to WebAssembly (Java, C#, Elm, Scala), it will also make it easier to interoperate with objects created by the host, which are often garbage collected, as well." [16]

4. Networking in Browser WebAssembly

API calling, for instance the JS API, often utilizes the memory of the wasm module to transfer information. In that way other APIs or system resources can be accessed.

4.1. Memory in WebAssembly

The memory is basically an array, whose indexes are the memory addresses. Memory is addressed via "[...]33 bit effective address that is the zero-based index [...]" [17] of the memory cell. Access to the memory contains, for instance, the ability to load or store bytes or grow the memory size to a certain amount of pages. (WebAssembly page size is 64Ki) [18]

4.2. How Modules Access APIs

A module does not have access to anything but its own memory and sandboxed runtime. To link modules together, "An embedder can instantiate multiple modules and use exports from one as imports to the other" [1]. Thus, a module can gain access to the functionality and memory of other modules.

APIs, for instance all browser APIs, libraries and devices, are nothing more than modules whose functionality can be accessed by importing them. Once imported, functions of the API can be called. However, WebAssembly still only supports arguments passed to these functions, that consist of the "[...] fundamental WebAssembly data types (i32 | i64 | f32 | f64 ; as the interface types and multi-value Wasm proposals get implemented, runtimes would also be able to exchange additional types, for example strings, and return more than one such value)." [18] The JS API, however, tries to convert strings into numbers so that they can be passed as arguments and interpreted [14].

Another way of passing arguments to the host runtime is by memory. Arguments are passed by invoking the module's function, passing a memory pointer. The function itself also returns a pointer to its memory so that information can be copied to and from both memories. In this case, both modules have to export parts of their memory to make it accessible. The function converting strings to integers and passing them through memory is also called Gluecode in the JS API [14].

4.3. Problems of Networking in WebAssembly

As mentioned in Section 3.2, networking directly from Wasm is hardly possible due to difficulties communicating with Web APIs. Another problem is, that even outside the browser an interface to directly support sockets for the Wasm module, making it possible to communicate with the web, still is not supported by most runtimes [19]. Even the WASI (Section 5) provides only little support for sockets. Being able to only call sockrecv, socksend, and sockshutdown is not enough to provide a proper TCP

doi: 10.2313/NET-NET-2022-07-1_13

connection. These methods are shown in the WASI snapshot of Wasmtime [19], but still unimplemented. However, there are multiple proposals to solve this problem. Hard design choices are currently keeping contributors from deciding on a proposal [20]. The two things that runtimes would have to do in order to make sockets available for modules are:

Firstly, define an API for networking (simply add a function definition to WASI);

Secondly, implement the functionality into their runtime [21].

4.4. Networking using the JS Interface

As mentioned in section 3.1, WebAssembly offers an API to JS. To circumvent the lack of bindings from Wasm to Web APIs, one could just leave all the networking to JavaScript. Although this method is a bit slower, due to the interfaces that have to be passed, there are not many more secure options.

In a browser, a Wasm module can be instantiated by with the method instantiateStreaming(). The instantiated Wasm module is a JS object and a WebAssembly.Memory object can be passed to it. JavaScript has full access to this memory, so passing arguments and reading return values from it is less complicated [22].

5. WASI

Until now, WebAssembly was only explained as an assembly language running in a browser environment. However, Wasm has proven its applicability and advantages, so developers tried to implement it outside the browser as well. Therefore an environment for Wasm to run on is needed. As mentioned in section 2.3, the runtime for Wasm allows a general and portable execution of modules as it creates an abtraction to specific operating system. Another ingredient that is needed is a way to communicate with the system. In a browser this was the Web API that offered an interface for system access. For outside the browser, this is exactly what WASI is. A Web Assembly System Interface [7].

5.1. WASI Structure

WASI is standardized to have one main core that most programs will need. For instance, this interface will provide access to functionality, such as random numbers, files or networking. Additional to this main WASI core, a system can decide whether it needs more special interfaces, such as crypto, 3D Graphics or processes. All in all WASI is supposed to be modular and every host can grant its Wasm programs only the functionality that he wants to provide.

If a module needs access to a specific interface, "the runtime that is running the code passes the wasi-core functions in as imports" [7]. This way the host system can decide for each module, if it wants to grant the requested interface.

6. WebAssembly Gateway Interface

As already mentioned Section 4.3 WASI itself does provide a snapshot for the most basic socket interfaces. These, however, are hardly implemented by runtimes.

The main idea of WAGI is to circumvent the lack of networking support in wasm and hand requests, that a wasm module wants to send, to a different program.

- The module writes the request to stdout,
- WAGI reads the request and sends it
- The response gets passed to the stdin file of the wasm module [21]

For this matter, "WAGI provides an HTTP server implementation that can dynamically load and execute WASM modules" [23]. The headers are accessed via environment variables, that are provided by WASI. Payloads are simply sent by using stdout and received in stdin.

6.1. WAGI Network Support

WAGI has to fill the networking gaps in WASI. To do so, WAGI provides a library for WASI that makes use of the idea explained above. The wasm runtime has to allow the module to use the specific API and mediate between the interface and the module. The WAGI HTTP server needs to have a list of all domains that the modules are allowed to access. An empty list indicates that modules are not allowed to request at all. There are also many possibilities to change the modules requests dynamically and execute different functions of the modules [23].

7. Conclusion

WebAssembly is an important innovation that improves the ability to bring all kinds of code to the web. Even outside the Browser it provides a stable and sophisticated language, that most programming languages profit compiling to, by means of security, portability and speed. Nevertheless, WebAssembly is not yet applicable for all use cases. Especially the network support lacks functionality. Even though WAGI seems to work around this problem, it can not provide a stable solution that is more than just experimental. (Users are warned to not use their approach other than just for experimenting) WebAssembly in the browser however, proved its applicability and advantage. This can be seen in many projects that bring, for instance, games and all kinds of software to the web. In the future, WebAssembly developers will try to implement a GC and solve many recent problems with that. Networking and all other kinds of support will follow. This will be the foundation for new APIs, bindings and a better compilation of programs dooming this project that emerged from asm.js, to success.

References

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, “Bringing the web up to speed with webassembly,” *ACM SIGPLAN Notices*, no. 6, p. 185–200, 2017. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3062341.3062363

[2] Emscripten Contributors, “About Emscripten,” https://emscripten.org/docs/introducing_emscripten/about_emscripten.html, 2015, [Online; accessed 16-November-2021],[Also good information on github].

doi: 10.2313/NET-NET-2022-07-1_13

[3] "Binary encoding," 2015, [Online; accessed 16-November-2021],[also on Github]. [Online]. Available: https://github.com/WebAssembly/design/blob/main/BinaryEncoding.md

[4] MDN contributors, "Understanding WebAssembly text format," https://developer.mozilla.org/en-US/docs/WebAssembly/Understanding_the_text_format, 2020, [Online; accessed 16-November-2021].

[5] Y. Shi, K. Casey, M. A. Ertl, and D. Gregg, "Virtual machine showdown," *ACM Transactions on Architecture and Code Optimization*, no. 4, p. 1–36, 2008.

[6] WebAssembly contributors, "Portability," https://webassembly.org/docs/portability/, 2021, [Online; accessed 16-November-2021].

[7] L. Clark, "Standardizing WASI: A system interface to run WebAssembly outside the web," 2019, [Online; accessed 16-November-2021]. [Online]. Available: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/

[8] WebAssembly contributors, "Security," https://webassembly.org/docs/security/, 2021, [Online; accessed 16-November-2021].

[9] Sander Tunge Aspøy, Helene Larsen, "Diversification for HotStuff through WebAssembly," Master's thesis, University of Stavanger, Stavanger, 2021.

[10] Aaron Turner (torch2424) with help from James Milner, Jonathan Beri (beriberikix) and Contributors. Additional input from Alex St. Louis, "Made with WebAssembly," https://madewithwebassembly.com/all-projects, 2019, [Online; accessed 16-November-2021].

[11] L. Clark, "Memory in webassembly (and why it's safer than you think)," 2017, [Online; accessed 16-November-2021]. [Online]. Available: https://hacks.mozilla.org/2017/07/memory-in-webassembly-and-why-its-safer-than-you-think/

[12] G. Royse. An introduction to webassembly. Youtube. [Online]. Available: https://www.youtube.com/watch?v=3sU557ZKjUs

[13] Chrome Developers, "Web APIs," 2021, [Online; accessed 16-November-2021]. [Online]. Available: https://developer.chrome.com/docs/extensions/api_other/

[14] L. Clark, "WebAssembly Interface Types: Interoperate with All the Things!" 2019, [Online; accessed 16-November-2021]. [Online]. Available: https://hacks.mozilla.org/2019/08/webassembly-interface-types/

[15] L. Wagner. Webassembly: status, webidl bindings, and roadmap. Youtube. [Online]. Available: https://www.youtube.com/watch?v=iwFsZdib8l4;https://www.w3.org/2018/12/games-workshop/slides/08-web-idl-bindings.pdf

[16] C. Eberhardt, "What Is WebAssembly?" https://www.oreilly.com/library/view/what-is-webassembly/9781492076902/ch04.html, 2019, [Online; accessed 16-November-2021].

[17] A. Rossberg, N. Z. An, B. Smith, and A. Yasui, "Instructions," 2017, [Online; accessed 16-November-2021],[also on Github]. [Online]. Available: https://webassembly.github.io/spec/core/syntax/instructions.html

[18] Radu Matei, "A practical guide to WebAssembly memory," https://radu-matei.com/blog/practical-guide-to-wasm-memory/#webassembly-memory, 2021, [Online; accessed 16-November-2021].

[19] WASI contributors, "Wasi_Snapshot_Preview1," https://github.com/bytecodealliance/wasmtime/blob/6db24fd08fa6f675e1b4ef818f8684602fd58730/crates/wasi-common/src/snapshots/wasi_snapshot_preview1.rs#L808-L828, 2020, [Online; accessed 16-November-2021].

[20] N. McCallum, "WASI: Expand Networking Functions," https://github.com/bytecodealliance/wasmtime/issues/70, 2019, [Online; accessed 16-November-2021].

[21] Radu Matei, "Wasi-Experimental-Http," https://radu-matei.com/blog/wagi-updates/, 2021, [Online; accessed 16-November-2021].

[22] MDN Contributors, "Using the WebAssembly JavaScript API," 2021, [Online; accessed 16-November-2021].

[23] Matt Butcher, Itowlson, "Architecture of WAGI," https://github.com/deislabs/wagi/blob/main/docs/architecture.md, 2021, [Online; accessed 16-November-2021].

doi: 10.2313/NET-NET-2022-07-1_13

NWCRG Closing Report

Aral Toksoy, Henning Stubbe*, Kilian Holzinger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: aral.toksoy@tum.de, stubbe@net.in.tum.de, holzinger@net.in.tum.de

***Abstract*—Network Coding Research Group (NWCRG) is concluding itself after 8 years of research. Throughout the years, as a part of the Internet Research Task Force (IRTF), the NWCRG has been exploring the concept of Network Coding, which is a networking technique, where a content of a packet is coded at a network node in a packet network. NWCRG also summarizes the research results and practical implementations related to Network Coding.**

This paper gives an overview and most important concepts of Network Coding and summarizes the efforts undertaken by the NWCRG in Network Coding.

***Index Terms*—Network Coding, NWCRG, IRFT, IEFT, IRSG, Internet Draft, Request For Comments**

1. Introduction

The history of Network Coding dates back to its initial introduction in the seminal paper [1] by Ahlswede et al. and the popularity of Network Coding has been ascending since. Taking notice of the potential power and benefits, researchers commenced to do a research about Network Coding and its possible practical implementations. Following several publications regarding and applications with the usage of Network Coding, the incentive to establish a research group in the IRTF was born.

Since its establishment, the NWCRG has accomplished numerous important works such as researching the principles of Network Coding, summarizing the existing publications and applications, and proposing new ideas. After having fulfilled their main objectives, the NWCRG is coming to an end in 2022 after 8 years of a journey, leaving various contributions to Network Coding.

In this following paper, the most important aspects of Network Coding will be covered and the Network Coding Research Group as well as the efforts undertaken by this group will be showcased. We firstly mention the theory and history behind the Network Coding and its benefits in Section 2. Moreover in Section 3, the history and the initial motivation of the NWCRG and their accomplishments will be explained. Finally, having analyzed the topics of interest, we draw a conclusion of the mentioned topics in Section 4.

2. Network Coding

Considering that Network Coding has had multiple definitions since its birth, it can be challenging to describe the concept with a particular interpretation.

The very first and most general definition of the Network Coding was framed in the seminal paper [1] in year 2000. The authors Ahlswede et al. stated that they will "refer to coding at a node in a network as network coding". The meaning behind coding in this paper was an arbitrary mapping of a data at intermediate network nodes.

According to Ho et al. in their book, Network Coding is a technique, where a content of a packet is coded at a network node in a packet network (a network, in which a data is broken down into packets). The network nodes can take multiple packets, combine them, and output a resulting packet to send it to the next node in the network [2].

2.1. History of Network Coding

In the seminal paper [1], the authors spoke of the potential power and benefits of Network Coding, however, did not disclose the design methods [3].

After 3 years from the first paper on Network Coding, Li, Yeung, and Cai published a new paper [4], which was more about the practical implementation of Network Coding. They showed that applying Network Coding on networks can rely on mathematical functions and suffices to achieve the optimum potential [3].

In the same year, Koetter and Medard issued a paper [5], in which an algebraic framework for analyzing coding approaches was introduced [3].

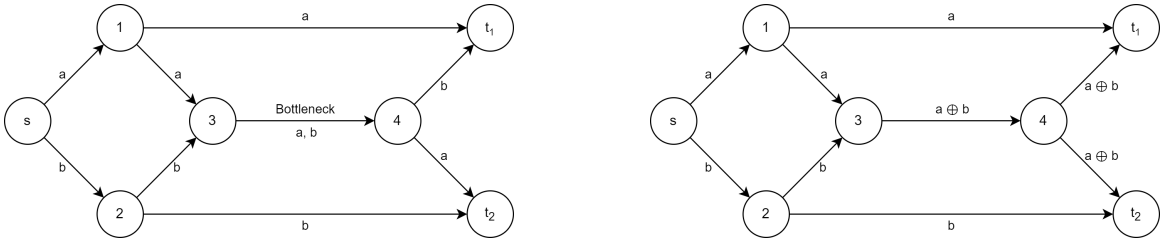
In 2005 and 2006, two important design algorithms were reported. In the first paper [6] from 2005, the authors, Jaggi et al., introduced polynomial time algorithms and randomized algorithms for Network Coding. In the second paper [7] from 2006, the authors Jaggi et al. expressed that randomly chosen network codes are also convenient for multicast networks [3].

2.2. Benefits of Network Coding

Especially in comparison with traditional routing networks, Network Coding can improve the throughput, complexity, robustness, latency, and security of a network. Therefore, in this part, we will go through the most important improvements along with the famous networks such as butterfly network and diamond network.

Increased throughput is one of the easiest-to-demonstrate benefits of Network Coding [2]. In the seminal paper [1], the potential gain in throughput is explained with a butterfly network. As in the paper, we will use s for the source, and t_1, t_2,..., t_n, the sinks of a graph. In both Figures 1a and 1b, the source s transmits two packets

doi: 10.2313/NET-NET-2022-07-1_14



(a) Routing with Multicast (b) Network Coding

Figure 1: Butterfly Network

(a, b) to the sink nodes t_1 and t_2 and the capacity of each edge is one packet per unit time.

With routing, the edge (3, 4) of Figure 1a poses a bottleneck, and it can carry either the packet a or b per unit time. Hence, it must be used twice so that both sinks can get the packets a and b. As seen in the Figure 1a, it is not possible to feature a multicast from one source node to two sink nodes simultaneously.

Through the medium of Network Coding, the node 3 in the Figure 1b is coded and takes both packets, a and b, combines them by taking the **xor** operation and sends out the new packet. The sink nodes t_1 and t_2 can decode the missing packets by taking the **xor** operation again with the two received packets. In this case, t_1 performs the **xor** operation between packets a and $a \oplus b$ and receives the missing packet, b. Likewise, t_2 performs the **xor** operation between packets b and $a \oplus b$ and receives the missing packet a. Thus, we see that using Network Coding in the butterfly network saves one transmission by coding an intermediate node and therefore increases the throughout [2].

Network Coding can reduce the complexity of a network in certain cases. In article [2], Ho and Lun state that although routing can gain comparable performance as Network Coding, Network Coding can reduce the complexity and consequently provides better performance in the fields such as Gossip-data dissemination protocol and Wireless Ad Hoc Network (WANET) [2], [8].

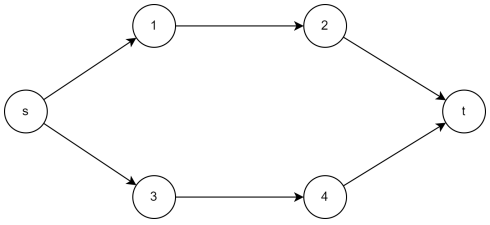


Figure 2: Diamond Network

Network Coding can have both advantages and disadvantages in terms of network security. On the one hand, Network Coding can make a network more vulnerable to attacks like Pollution Attacks or Byzantine Attacks. Mixing packets at the intermediate nodes can result in packet pollution and one polluted packet can easily spread the pollution to many other packets. Considering that this pollution attack is one of the dangerous attacks against a network, it may cause a failure of the decoding at the sink nodes [9].

However, on the other hand, Network Coding can ensure a secure communication within a network by preventing eavesdropping. For example, in the Figure 2, the source node s wants to transmit a packet p to the sink node t, but it is aware that one of the 4 intermediate nodes in the network is malicious and operated by an eavesdropper. With routing, there is a 1/2 chance that the eavesdropper obtains the packet. With Network Coding, s can split the packet s into 4 equal packets p_1, p_2, p_3 and p_4 and then combine these packets again by taking the **xor** operation:

$$c_1 = p_1 \oplus p_2, c_2 = p_3 \oplus p_4 \quad (1)$$

$$c_3 = p_1 \oplus p_4, c_4 = p_2 \oplus p_3 \quad (2)$$

Then, s sends these combined packets c_1 and c_2 over node 1 and the other packets c_3 and c_4 over node 2. If the malicious node cannot calculate the content of at least one coded packet, decoding the packet p is impossible [2], [8].

3. Network Coding Research Group

Network Coding Research Group (also known as Coding for efficient NetWork Communications Research Group) is one of the 14 active and chartered research groups of the IRTF.

Before going into the particulars of the NWCRG, we would like to refer briefly to the IRTF and its significance.

3.1. Internet Research Task Force

IRTF is an organization that is administered by the IRTF-Chair in deliberation with the Internet Research Steering Group (IRSG) and targets research-issues connected to the Internet such as Internet protocols, applications, architecture, and technology, by establishing long term research groups working on these fields [10]. The roles of research groups can be listed as [11]:

- Finding new ideas about their field
- Exploring new academic and industrial areas with global Internet potential
- Expert-supporting the Internet Engineering Task Force (IETF)

On the contrary, the parallel organization IETF focuses on issues of engineering and standards making in the short term [10].

The IRTF also organizes the Applied Networking Research Workshop and the Applied Networking Research Prize in collaboration with the Association for Computing

doi: 10.2313/NET-NET-2022-07-1_14

Machinery (ACM) and Internet Society (ISOC), where the applied networking research results are reviewed and discussed by the researchers, vendors, and many other operatives in the field [12].

3.2. History of NWCRG

Following the first introduction of the concept of Network Coding in the article [1], the interest in the field increased consistently. After taking action on practical implementations and design algorithms throughout the years, the involvement in pursuing an IRTF activity came on stage.

In November 2012, researchers from Lincoln Laboratory, MIT, Caltech, and many other institutions from all around the globe were invited to constitute an IRTF research group led by Victor Firoiu from BAE Systems, and Brian Adamson from U.S. Naval Research Laboratory. By the end of 2012, the research group was formed and had planned a meeting at the IETF-86 Meeting in Orlando in order to introduce the research group's objectives and plans for the future. Succeeding the primary meeting, the group decided to plan a second meeting in the next IETF-87 Meeting in Berlin. Having reintroduced the research group and its objectives in IETF 87 and IETF 88 Meetings, the IETF chair officially chartered the research group, and the Internet Architecture Board (IAB) approved the charting on the 13^{th} of November 2013 [13].

One year after laying the foundation for a research group, the Network Coding Research Group was officially one of the chartered research groups of the IRTF.

3.3. Initial Motivation

As industrial and commercial applications enhanced their interest in Network Coding after few years of research, the Network Coding Research Group was established as mentioned. According to the presentation of Firoiu and Adamson in their first IETF Meeting, one of the initial motivations of the NWCRG was to analyze the research advancements, proved performance improvements and the practical algorithms in the publications [14]. For instance, analyzing the achievements of multicast with Network Coding on Max-flow Min-cut problem in the seminal paper of Ahlswede et al. [1] as well as analyzing the coding schemes for reliable communication from the article of Lun et al. [15].

Additionally, one of the initial motivations of the research group was to do a research regarding the practical applications of Network Coding as the studies on Network Coding expanded and developed over the years and began to appear on manifold platforms [14].

3.4. Objectives

Every research group affiliated with the IRTF is founded based on a peculiar requirement and has a specific objective. As declared in the charter [16], the main goal of the Network Coding Research Group is to analyze principles and utilities of Network Coding in order to improve the Internet communication. NWCRG examines the research results on Network Coding and aims to advance its practical applications. In addition, the group also focuses on the existing practical implementations and targets to achieve the standardization of the Network Coding enabled communication [16].

3.5. Interest Areas of the NWCRG

As stated above, the two main interest areas of the Network Coding Research Group are Network Coding Research and Practical Applications of Network Coding. In these sections, we will see the primary subjects, in which the NWCRG is interested.

NWCRG is mainly devoted to the following topics in Network Coding Research [16]:

- Performance and efficiency: Analyzing performance improvements, computational complexity of Network Coding, trade-offs between different Network Coding techniques.
- Security, privacy, and robustness: Evaluating the advantages and disadvantages of Network Coding in network security, analyzing the interactivity of Network Coding and encryption.
- Application Layer: Exploring the interactions between Network Coding and application-specific Coding.
- Data Link Layer: Searching for the interaction between Network Coding and data link protocols such as optical, wireless, and satellite links.
- Costs of Network Coding: Determining the ways to price services, for instance, network usage or information rate.

NWCRG is mainly interested in the following matters in Network Coding practical application [16]:

- Architectural Considerations: Analyzing different design techniques and requirements for extensive networks.
- End-to-end vs. hop-by-hop: Evaluating the two different transport principle.
- Intra-flow and inter-flow: Researching the two different protocols Intra-flow Network Coding and Inter-flow Network Coding.
- Service Paradigms: Analyzing the service paradigms such as best effort and time bounded utility.
- Encoding – Decoding algorithms, packet formats: Examining the benefits of Network Coding in common encoding and decoding algorithms and packet formats.

3.6. Achievements

Since its establishment in 2013, the Network Coding Research Group has made many contributions to Network Coding by encouraging studies on Network Coding and improving network performance, developing codes, and coding libraries, offering new protocols to promote the usage of Network Coding in existing and future systems. The group published many Internet-Drafts (I-Ds) and Request For Comments (RFC) over the years [17].

Before going into the details of the most important achievements and proposed protocols of the NWCRG, we

will briefly clarify the terms Internet-Drafts (I-Ds) and Request For Comments (RFC).

An Internet-Draft is a short-lived working document of the Internet Engineering Task Force (IETF). It is the primary input with technical standards and research findings which then may be approved as a Request For Comments. Since I-Ds are ongoing documents and do not embody any formal status, they should not be cited or acknowledged as authoritative sources [18], [19].

On the other hand, a Request For Comments is a publication with technical specifications and organizational notes, which poses more formality. Furthermore, an RFC can have different IETF statuses based on the maturity level [18], [20].

Immediately below, we will introduce one Internet-Draft and one RFC of the Network Coding Research Group.

Network Coding for Content-Centric Networking / Named Data Networking: Considerations and Challenges is one of the three Internet-Drafts of the NWCRG. The I-D was sent to Internet Research Steering Group (IRSG) in March 2021 and currently, this group is still in the process of approval and taking a poll on whether the document fulfills the requirements to be published [21].

The first version of the I-D was approved in October 2018. In this document, the authors Matsuzono, Asaeda and Westphal summarize the current research in Network Coding for Content-Centric Networking (CCN) Named Data Networking (NDN) and technical issues and challenges when applying Network Coding to the CCN and NDN such as content naming, transport, congestion control and security. Matsuzono et al. state that, the application of Network Coding in CCN and NDN can help with large-scale content/information dissemination effectively. Combining Network Coding and CCN/NDN may also cause security issues such as malevolently requesting or injecting network packets and thereby resulting in amplification attacks [21].

The RFC *Taxonomy of Coding Techniques for Efficient Network Communications* [22] is the first RFC of the Network Coding Research Group which was approved by the IRSG and was published in 2018. The purpose behind this RFC was to create a Network Coding terminology which would assist the future research on Network Coding, and instead of proposing specific solutions offering the common Network Coding concepts, constructs and set of terms. It was the first document to be published about the taxonomy of Network Coding and 14 researchers from the NWCRG contributed to this RFC.

In this RFC, the general definitions and concepts of the Network Coding such as Packet Erasure Channel, End-to-End Coding, Source Node, Coding Node and surely the Network Coding are clarified. The authors refer to Network Coding as: "A system where coding can be performed at the source as well as at intermediate forwarding nodes (all or a subset of them)." [22]. The different coding types and the technical details are also discussed in the document.

In substance, the RFC *Taxonomy of Coding Techniques for Efficient Network Communications* summarizes the recommended terminology for Network Coding and the most common definitions of Network Coding that many of the researchers use and come across while researching or developing practical applications [22].

3.7. Current Situation

After 8 years of doing research, developing practical applications, positing new questions, and publishing many I-Ds and RFCs, the Network Coding Research Group organized their last meeting at the online IETF 111, led by the co-chairs Marie-Jose Montpetit and Vincent Roca.

The group has 3 active I-Ds at the moment. The I-Ds, *Coding and congestion control in transport* and *Network Coding for CCN / NDN: Considerations and Challenges*, are sent to the IRSG and the latter is currently in IRSG poll and will be published according to the results of the poll. The third I-D, *BATS Coding Scheme for Multi-hop Data Transport*, which analyzes and discusses the BATS coding schemes for multi-hop networks communications, has not been sent to the IRSG [21]–[23].

After sending all I-Ds to the IRSG, and publishing these I-Ds as RFCs, the Network Coding Research Group will be closed in 2022, after 9 years of the establishment.

4. Conclusion

To conclude this paper, we have provided an overview of Network Coding theory and its most important benefits. We have also shown that the Network Coding Research Group has complied with its initial motivations and accomplished their fundamental objectives by making research concerning the Network Coding, summarizing publications and applications, and proposing newfangled ideas.

Furthermore, the prospective individual research and implementations on Network Coding from the researchers of the NWCRG can be of interest in the future.

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] T. Ho and D. Lun, *Network Coding: An Introduction*. USA: Cambridge University Press, 2008.

[3] M. Effros, M. Médard, and R. Koetter, "A Brief History of Network Coding," https://www.scientificamerican.com/article/history-of-network-coding/, 2007.

[4] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[5] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[6] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial Time Algorithms for Multicast Network Code Construction," *IEEE Transactions on Information Theory*, vol. 51, pp. 1973 – 1982, 07 2005.

[7] S. Jaggi, "Design and Analysis of Network Codes," Ph.D. dissertation, California Institute of Technology, USA, 2005.

[8] G. Carle, S. Günther, W. Utschick, and M. Riemensberger, "Network Coding," https://www.net.in.tum.de/pub/nc2014/slides.pdf, 2014.

[9] P. Ostovari and J. Wu, *Toward Network Coding for Cyber-Physical Systems: Security Challenges and Applications*. John Wiley & Sons, Ltd, 2017, ch. 11, pp. 223–242. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119226079.ch11

[10] "Internet Research Task Force," https://irtf.org/, 2021.

[11] A. Mankin, "IRTF Overview," https://datatracker.ietf.org/meeting/99/materials/slides-99-edu-sessd-irtf-overview-01.pdf, p. 15, 2017.

[12] "Applied Networking Research Workshop," https://irtf.org/anrw/, 2021.

[13] "IETF Mail Archive," https://mailarchive.ietf.org/arch/browse/nwcrg/, 2021.

[14] B. Adamson and V. Firoiu, "Introduction and Overview Network Coding Research Group," https://www.ietf.org/proceedings/86/slides/slides-86-nwcrg-0.pdf, pp. 4, 5, 2013.

[15] D. Lun, M. Médard, R. Kötter, and M. Effros, "On coding for reliable communication over packet networks," *Physical Communication*, vol. 1, pp. 3–20, 09 2005.

[16] "Charter - Network Coding Research Group," https://datatracker.ietf.org/doc/charter-irtf-nwcrg/, 2014.

[17] M.-J. Montpetit and V. Roca, "Coding for efficient NetWork Communications Research Group (NWCRG)," https://datatracker.ietf.org/meeting/111/materials/slides-111-nwcrg-00-nwcrg-ietf111-chairs-00/, 2021.

[18] "What are Internet Drafts and Requests for Comments (RFCs)?" https://kb.iu.edu/d/agkj, 2018.

[19] "Internet-Drafts," https://www.ietf.org/standards/ids/, 2021.

[20] "RFCs," https://www.ietf.org/standards/rfcs/, 2021.

[21] K. Matsuzono, H. Asaeda, and C. Westphal, "Network Coding for Content-Centric Networking / Named Data Networking: Considerations and Challenges," Internet Engineering Task Force, Internet-Draft draft-irtf-nwcrg-nwc-ccn-reqs-08, Nov. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-nwcrg-nwc-ccn-reqs-08

[22] B. Adamson, C. Adjih, J. Bilbao, V. Firoiu, F. Fitzek, S. A. M. Ghanem, E. Lochin, A. Masucci, M.-J. Montpetit, M. V. Pedersen, G. Peralta, V. Roca, P. Saxena, and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications," RFC 8406, Jun. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8406.txt

[23] S. Yang, X. Huang, R. W. Yeung, and D. J. K. Zao, "BATS Coding Scheme for Multi-hop Data Transport," Internet Engineering Task Force, Internet-Draft draft-irtf-nwcrg-bats-03, Dec. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-nwcrg-bats-03

ISBN 978-3-937201-75-7
DOI 10.2313/NET-NET-2022-07-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)