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

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

Summer Semester 2021

Munich, Germany

Editors	Georg Carle, Stephan Günther, Benedikt Jaeger
Publisher	Chair of Network Architectures and Services

Network Architectures and Services NET 2022-01-1

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

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

Summer Semester 2021

Munich, March 5, 2021 – August 8, 2021

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET 2022-01-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Summer Semester 2021

Editors:

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

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

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

Cataloging-in-Publication Data

Seminar IITM SS 21
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, March 5, 2021 – August 8, 2021
ISBN: 978-3-937201-73-3

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2022-01-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2022-01-1
Series Editor: Georg Carle, Technical University of Munich, Germany

Preface

We are pleased to present you the proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM) during the Summer Semester 2021. 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 afterwards 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 arwards where given to Christopher Pfefferle with the paper *IEEE 802.1Qcr Asynchronous Traffic Shaping with Linux Traffic Control* and Paul Schaaf with the paper *Analysis of Proof of Stake flavors with regards to The Scalability Trilemma* .

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, November 2021

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

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

Juliane Aulbach (aulbach@net.in.tum.de)
Technical University of Munich

Stephan Günther (guenther@tum.de)
Technical University of Munich

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

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

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

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

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

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

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

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

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

Seminar Homepage

`https://net.in.tum.de/teaching/ss21/seminars/`

Contents

Simulation of WiFi Mesh Networks with Mobile Nodes

Sebastian Keller, Jonas Andre*, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: sebastian2.keller@tum.de, andre@net.in.tum.de, wiedner@net.in.tum.de

***Abstract*—Investigation of WiFi networks can be performed with real testbeds or with the help of simulation software. Real testbeds are expensive and it is difficult to obtain consistent and reproducible test results. The results of simulated tests are only comparable to the real world to some extend. The advantages of simulators are lower costs and reproducibility. Simulators are particularly suitable for investigating WiFi meshes with mobile nodes, as these are difficult to implement with real testbeds.**

There are different software solutions for network visualization available. This paper describes two network simulators, *ns-3* and *OMNET++*, which are compared in terms of their capabilities to simulate WiFi mesh networks with mobile nodes. The comparison shows that *ns-3* is better when considering performance and customization, while *OMNET++* offers more features related to mesh networks with mobile nodes, e.g. extensive visualizations. Still, both provide no implementation for the latest WiFi standards.

***Index Terms*—Wifi-Networks, *ns-3*, *OMNET++*, Simulations, Wireless Mesh Network**

1. Introduction

Wireless communication is a key-technology for next generation devices of any kind. Increasing bandwidth and range of WiFi allows more and more applications to switch from wired connections to wireless ones. New medical applications use WiFi for in-body sensors [1], autonomous driving requires real-time communication between multiple road users, and wireless communication enables new features for drone swarms [2].

Simulation of networks is an attempt to imitate real properties of digital communications. They are used to speed up the development process of new WiFi features as well as to determine anomalies earlier. Simulation of wired networks is less complex, e.g. influence of the environment is less dominant and the complexity of the hardware is less. WiFi networks suffer of interference, connection losses, and retransmissions. Thus, simulation of WiFi networks is more difficult than simulation of wired networks. The data collected from a simulated wireless network can only be compared to real-world network in some extend.

Another approach to investigate WiFi networks are real testbeds. These testbeds are rather expensive, the environment needs to be screened from the surrounding environment to avoid inferences of the surrounding. Thus, the results are less reproduceable than results from simulations.

When it comes to the simulation of dynamic mesh networks, additional problems arise [2]:

1) Link-quality is not constant over time due to the limited range and moving links.
2) Links can break because nodes might move out of range. The reachable neighbours of each node change over time.
3) Routing may change, e.g. because a link loses the original communication partner, but another link is in range, which can be used for communication.

Figure 1 shows an example of a flying Ad-Hoc network which illustrates the stated problems. The WiFi coverage of each drone is illustrated by the red circles. While the drones are flying their reachable neighbours change. It must be assured, that at least one drone can connect with the gateway drone, which connects the whole swarm with the internet using a cellular connection. This illustrates how an area without cellular coverage can be covered with a WiFi network in case of an emergency.

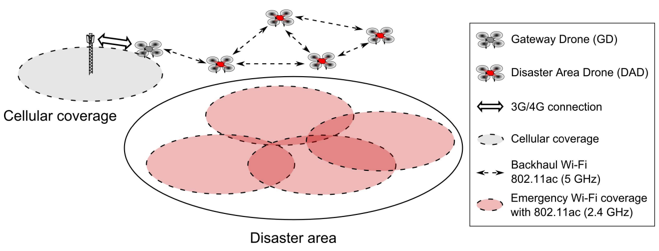

Cellular coverage
Disaster area
Gateway Drone (GD)
Disaster Area Drone (DAD)
3G/4G connection
Cellular coverage
Backhaul Wi-Fi 802.11ac (5 GHz)
Emergency Wi-Fi coverage with 802.11ac (2.4 GHz)

Figure 1: Flying Ad-Hoc network example [2].

The development of new wireless communication standards causes rapid progress in terms of transmission rates and new features. Thus, there exist no simulation which is capable of simulating all standards. The youngest simulation software is *ns-3*. It is the successor of *ns-2* with the focus on scalability and performance. *ns-3* is open source and it has a huge community which allows fast development of new models. *OMNET++* is a commercial solution for network simulation which is free for non-commercial use. There are other simulation software with different focus, e.g. *OPNet*, *Castalia*, *Qualnet*, *Tetcos NetSim*, *OpenSim*, *MIMIC Wireless Simulator*, and the *Wireless Sensor Networks Simulation* Extension for *Matlab*.

In Chapter 2 related work on the topic of WiFi mesh virtualization is presented. The concepts behind *ns-3* and *OMNET++* are described in Chapter 3. Afterwards the benefits and drawbacks of each simulation software are compared in Chapter 4. In the last Chapter a conclusion is drawn containing a short summary of the benefits and drawbacks of each.

2. Related work

The limitations of *ns-3* are explained in detail in "Network Simulation and its Limitations" [3]. According to their research, wireless network simulation provides only limited credibility and scalability. Credibility can be increased by limiting the simulation results only to certain aspects of the network, performing regressions tests, reusing existing and tested code, and by comparing the simulation results with the results from a real testbed.

Scalability is another issue for real testbeds, as the testbed is going to be more costly with increasing number of nodes. Simulations have also limited scalability. This limit can be increased by parallel computing or distributed network simulation.

Other publications present ways to extend the current available models of *ns-3* with features, which are necessary to virtualize WiFi Mesh Networks. Hany Assasa et al. show a solution for beamforming in [4]. They extended the existing *ns-3* model for *IEEE 802.11ad* to support multiple antenna beamforming.

As WiFi mesh virtualization has multiple usecases, there exist several publications which focus on specific applications like flying Ad-Hoc network systems [5] or Wireless Body Area Networks (WBAN) [6]. Dmitrii Dugaev and Eduard Siemens demonstrate the use of WiFi meshes to enable communication between multiple flying drones in [5]. The key challenges of this application are the points 1, 2, and 3 stated in Chapter 1. WBANs are used to monitor body measurements like electrocardiograms (ECG) or electroencephalograms (EEG). Beom-Su Kim et. al. use *ns-3* to build a realistic WBAN simulation system [6].

The comparison of the virtualization with the real-world is necessary to validate simulation models. This has been done by Dmitrii Dugaev and Eduard Siemens in [7]. They compared experiments in *ns-3* with a real testbed. They conclude that the physical, interference, and channel models are sufficiently accurate and they state that it can be used to evaluate performance parameters of "different wireless mesh networks with various topologies" [7]. Also, the *ns-3* model for hybrid routing schemes and link establishing algorithms can be used for this purpose.

3. Concepts

As already mentioned in Chapter 1, there exist different network simulators. Several studies [8]–[12] compare different network simulators to simulate wireless networks. The following chapters will focus on *OMNET++* and *ns-3* as they both provide good performance, are actively maintained, and free for academic use.

3.1. ns-3

ns-3 is a discrete-event network simulation tool, i.e. each step in simulation time is assigned to every active event, events are triggered consecutively in discrete steps. It is published under the *GNU GPLv2* license and thus the source code is available. The core of *ns-3* is written in *C++* while the scripting interface is written in *Python*. It is developed for research and educational purposes. In comparison to its predecessor *ns-2*, *ns-3* is developed with focus on scalability and performance.

ns-3 has modular structure containing the following main features [13]:

1) **Nodes**: a communication point, e.g. router, smartphone
2) **Channels**: interconnect multiple nodes, e.g. *PointToPointChannel* or *WifiChannel*
3) **NetDevices**: represent a physical interface on a node, e.g. an Ethernet interface
4) **Packets**: packets are sent over channels using NetDevices
5) **Sockets and Applications**: user defined processes that generate packets

These components are used to define the network topology. Figure 2 shows a schematic configuration in *ns-3*, which illustrates the structure of a *ns-3* model. For simulation of WiFi networks, the channel is a *WifiChannel* and the nodes are WiFi clients with implemented applications, which can communicate with other nodes using WiFi.

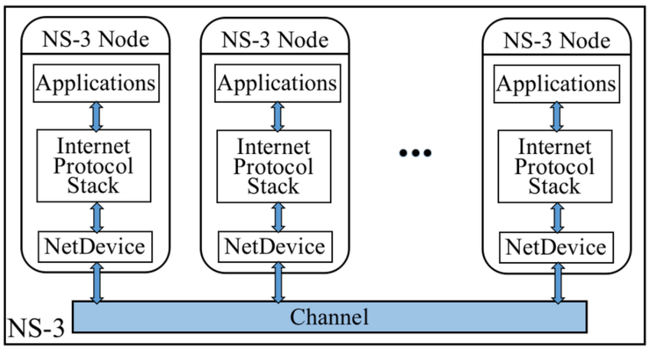


Figure 2: data flow model in *ns-3* simulation at a high-level [14].

The simulation needs to be initialized with events that will trigger the creation of further events. While the simulation is running, it is necessary that test results can be collected. *ns-3* includes a tracing subsystem which allows to measure and log data in a flexible way. The tracing subsystem can save the data collections in common data formats like *pcap*. This makes analysation with third-party software like *Wireshark* [15] possible. The simulation is terminated by either a specified simulation time, or if the list of upcoming events is empty.

ns-3 has a large community, which continuously improves existing models and adds new models. This allows to simulate the latest standards. Thus, there is a large model library available for *ns-3*.

ns-3 itself does not provide any graphical user interface. This might be an issue on the first glance, but nevertheless it has proven to be comprehensible and easy to use. There exist several third-party visualization tools like *NetAnim* to animate tracing data, *Gnuplot* for general visualizations or the already mentioned *Wireshark*.

3.2. OMNET++

OMNET++ is a discrete-event simulator written in *C++*, just like *ns-3*. It is open source, but only free for non-commercial purposes. The commercial version of *OMNET++* is *OMNEST*. It was not originally developed as a network simulator, but as a general purpose discrete event simulator. In contrast to *ns-3*, *OMNET++* provides a graphical user interface called *OMNET++ IDE*. The *OMNET++ IDE* can be used to create NED files, which are used to define components. This includes the definition of modules, networks, and connections. *OMNET++ IDE* also includes a source editor as an alternative.

The functionality of the modules is implemented in *C++*. Each module is represented by a class which handles the initialization of a module and the overall behaviour of the module.

This network definition consisting of the NED files and C++ classes is then used to run the simulation. *OMNET++ IDE* can be used to visualize the progress of the current simulation in detail.

In contrast to *ns-3*, *OMNET++* provides many integrated visualization tools like the *TransportRouteVisualizer* to visualize traffic passing through the transport layers of multiple endpoints, the *NetworkRouteVisualizer* to visualize network layer traffic, *Ieee80211Visualizer* for *IEEE 802.11* networks and *MobilityVisualizer* to visualize the mobility of nodes. Figure 3 shows the network *Tictoc14* with 6 nodes during a TicToc simulation visualized with *OMNET++*. Packets are illustrated with a red dot, which moves between the nodes during simulation. This example counts the number of received and sent packets.

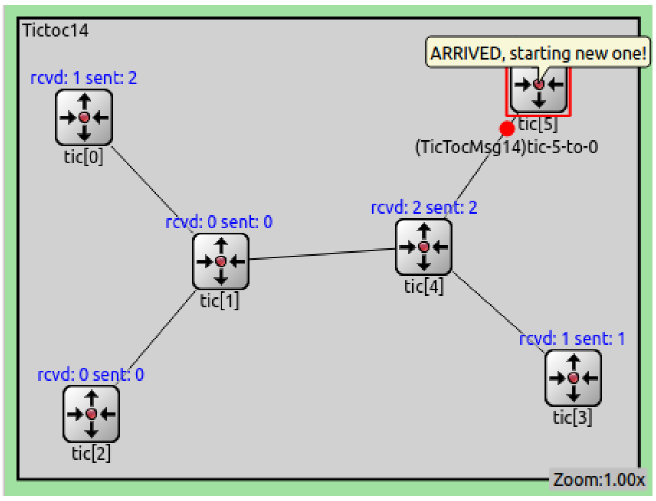


Figure 3: Example network visualization in *OMNET++* [16].

4. Comparison

The key challenges for the simulation of Wifi Mesh networks with dynamic link-quality are:

1) support for multiple antenna beamforming
2) support for Ad-Hoc routing
3) simulation of node-movement, i.e. their dynamically changing link-quality
4) simulation of packet-loss and retransmissions
5) simulation of interference

The capabilities of *ns-3* and *OMNET++* with respect to these requirements are outlined in this Chapter.

4.1. Multiple antenna beamforming

Multiple antenna beamforming is a technique that improves the signal quality with the same energy requirements. This is achieved by directional signal transmissions. The *INET Framework* of *OMNET++* supports various directional antenna, transmitters, and receivers. The *RadioVisualizer* module of *OMNET++* includes visualizations of all available antenna models. Nodes with multiple different antennas can be modeled by adding multiple wireless interfaces and assigning different antenna models to them.

As already mentioned in Chapter 2 *ns-3* has no built-in support for multiple antenna beamforming, but this functionality has been implemented in [4].

4.2. Ad-Hoc routing

The *IEEE 802.11s* standard is an extension of the *IEEE 802.11 MAC* standard which was developed to support Ad-Hoc networks with minimum hardware requirements and reduced energy consumption. The *IEEE 802.11* specification released in 2012 also includes support for mesh routing.

OMNET++ supports Ad-Hoc routing including several routing protocols *AODV*, *DSDV*, *DYMO*, and *GPSR*. Its *802.11* models includes the *Ieee80211MgmtAdhoc* management component for Ad-Hoc mode stations [16].

ns-3 does support the Ad-Hoc routing protocols *AODV*, *DSDV*, *DSR*, and *OLSR* [17]. The different protocol implementations are compared in [18]. The results showed that *OLSR* has the best performance. *ns-3* supports *802.11s* besides many other WiFi specifications, but still lacking the latest *IEEE 802.11ay* standard.

4.3. Node mobility

OMNET++ offers built-in mobility models including stationary, deterministic, trace-based, stochastic, and combined models. By default the antenna mobility model uses the same mobility model as the node itself, but it is also possible to define independent models. This allows e.g. to model a vehicle with multiple directional antennas located at different positions in the vehicle. The *MobilityVisualizer* allows to visualize the motion of mobile nodes.

ns-3 includes mobility models to define the position, velocity, and acceleration of nodes. It does not support movement along the Z dimension currently. An approach to extend the mobility model to three dimensions is presented in [19].

4.4. Packet-loss and retransmissions

The simulation of packet-losses and retransmission is necessary to identify poor connectivity, overloaded nodes, or misconfigured nodes.

Simulator	*OMNET++*	*ns-3*
Open Source	Mostly	Yes
Free for commercial use	No	Yes
Wireless support	Yes	Yes
Scalability	Medium	Good
Performance	Medium	Good
Documentation	Great	Good
Visualization	Included/Good	Third-party/Medium
Multi antenna beamforming	Yes	Possible
Ad-Hoc support	Yes	Yes
Mobility support	Yes	Yes
Packet loss/retransmissions	Yes	Yes

TABLE 1: *ns-3* and *OMNET++* Comparison

OMNET++ supports packet drops and retransmissions including a visualization for packet drops with the *PacketDropVisualizer* module.

ns-3 does support packet losses and retransmissions, too. It has an included *PacketLossCounter* class, which can be used to count the number of lost packets.

4.5. Interference

Interference in WiFi signals can have many reasons. It is mostly caused by multiple overlapping WiFi signals using the same channel. This can cause slower networking speed, higher latencies, retransmissions, interrupted connections, and the inability to connect to a WiFi network.

OMNET++ offers debugging tools to investigate interference including many different visualizations.

ns-3 does also support interference. The *InterferenceHelper* class helps to trace many information relevant to investigate the interference.

5. Conclusion and future work

A comparison of *ns-3* and *OMNET++* with respect to WiFi meshes with mobile nodes is presented in this paper. Key features a simulator needs to support are outlined. *ns-3* offers slightly better performance. One drawback is the missing visualization, but it is still possible to use third-party tools. The possibility to create common tracing files that can be viewed in third-party tools like *Wireshark* overcomes this issue only to some extent. In general *ns-3* can be considered to be a more low-level simulation than *OMNET++*.

OMNET++ has a good documentation including well documented examples. The examples in the documentation of *ns-3* are less user-friendly, but the documentation of the modules and classes is detailed. One key-feature is the included visualization tool. Table 1 lists a comparison of the most relevant features. As *OMNET++* is only free for non-commercial use, *ns-3* might be the better choice in commercial applications. Both support the simulation of wireless communication, whilst *ns-3* supports more WiFi standards in a more accurate implementation. Ad-Hoc networks can be simulated by both tools. Simulation of multi antenna beamforming, which might be necessary for Ad-Hoc networks, is only integrated in *OMNET++*. The implementation of [4] can be used to add support for multi antenna beamforming in *ns-3*, too. *ns-3* is better in terms of scalability. It provides less memory consumption and simulation time than *OMNET++*.

This paper focuses more on a raw feature comparison of the simulation tools than on the test results the simulators produce. Future work needs to investigate the comparability of simulation results with test results gained from a real testbed.

References

[1] D. Rathee, S. Rangi, P. Chakarvarti, and V. Singh, “Recent trends in wireless body area network (wban) research and cognition based adaptive wban architecture for healthcare,” *Health Technol.*, pp. 1–6, 05 2014.

[2] J. Sae, S. F. Yunas, and J. Lempiainen, “Coverage aspects of temporary lap network,” in *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, 2016, pp. 1–4.

[3] S. Rampfl, “Network Simulation and its Limitations,” *Lehrstuhl Netzarchitekturen und Netzdienste, Fakultät für Informatik, Technische Universität München*, 2013.

[4] H. Assasa, J. Widmer, T. Ropitault, and N. Golmie, “Enhancing the ns-3 ieee 802.11ad model fidelity: Beam codebooks, multi-antenna beamformig training, and quasi-deterministic mmwave channel,” in *Proceedings of the 2019 Workshop on Ns-3*, ser. WNS3 2019. Association for Computing Machinery, 2019, p. 33–40. [Online]. Available: https://doi.org/10.1145/3321349.3321354

[5] E. Erkalkan, V. Topuz, and A. Buldu, “Heuristic algorithms testbed for flying ad-hoc network systems,” 2020, pp. 1–4.

[6] B.-S. Kim, T.-E. Sung, and K.-I. Kim, “An NS-3 Implementation and Experimental Performance Analysis of IEEE 802.15.6 Standard under Different Deployment Scenarios,” *Inernational Journal of Environmental Research and Public Health*, 2020.

[7] D. Dugaev and E. Siemens, “A wireless mesh network ns-3 simulation model: Implementation and performance comparison with a real test-bed,” 2014. [Online]. Available: http://dx.doi.org/10.25673/5633

[8] A. ur Rehman Khan, S. M. Bilal, and M. Othman, “A performance comparison of network simulators for wireless networks,” *CoRR*, vol. abs/1307.4129, 2013, [Online; accessed 21-April-2021]. [Online]. Available: http://arxiv.org/abs/1307.4129

[9] S. Mehta, N. Ullah, M. H. Kabir, M. N. Sultana, and K. S. Kwak, “A case study of networks simulation tools for wireless networks,” in *2009 Third Asia International Conference on Modelling Simulation*, 2009, pp. 661–666.

[10] V. Venkataramanan and S. Lakshmi, “A case study of various wireless network simulation tools,” *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 10, no. 2, 2018, [Online; accessed 21-April-2021]. [Online]. Available: https://www.ijcnis.org/index.php/ijcnis/article/view/3261

[11] N. Garg, “Network Simulators: A Case Study,” *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)*, vol. 5, 1 2015.

[12] A. S. Toor and A. K. Jain, “A survey on wireless network simulators,” *Bulletin of Electrical Engineering and Informatics*, vol. 6, no. 1, pp. 62–69, 2017, [Online; accessed 21-April-2021]. [Online]. Available: https://beei.org/index.php/EEI/article/view/568

[13] ns 3 Project, “ns3 Softare Architecture,” https://www.nsnam.org/docs/architecture.pdf, 2007, [Online; accessed 26-March-2021].

[14] B. Ren, “High Fidelity Experiment Platform for Mobile Networks,” https://www.researchgate.net/figure/Data-flow-model-in-NS-3-Simulation-at-a-High-level_fig3_322515716, [Online; accessed 26-March-2021].

[15] “Wireshark Website,” https://www.wireshark.org/, [Online; accessed 23-April-2021].

[16] “OMNET++ Documentation,” https://docs.omnetpp.org/tutorials/tictoc/part4/, [Online; accessed 26-March-2021].

[17] “ns-3 Documentation,” https://www.nsnam.org/docs/release/3.17, [Online; accessed 23-April-2021].

[18] M. Ikeda, M. Hiyama, E. Kulla, and L. Barolli, “Mobile ad-hoc network routing protocols performance evaluation using ns-3 simulator,” in *2011 Third International Conference on Intelligent Networking and Collaborative Systems*, 2011, pp. 14–20.

[19] D. Broyles, A. Jabbar, and J. Sterbenz, “Design and analysis of a 3–d gauss-markov mobility model for highly-dynamic airborne networks,” *International Telemetering Conference*, 01 2010.

Challenges with BGPSec

Jan Oesterle, Holger Kinkelin*, Filip Rezabek*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge25goc@mytum.de, kinkelin@net.in.tum.de, frezabek@net.in.tum.de

Abstract—**BGP serves as the standard inter-domain routing protocol. It exchanges Network Layer Reachability information between Autonomous Systems and by this ensures connectivity across the Internet. At the time BGP was introduced, there were no security concerns. The inadequate security led to numerous attacks on the Internet, the paper covers. The lack of security resulted in multiple different attempts to fix this issue. One of these attempts is BGPSec. This paper explains this extension to BGP and discusses the degree of security it offers. Because additional security comes with an additional cost, this paper analyzes the deployment issues that exist. In conclusion, it was found that BGPSec is a good start as it solves some existing vulnerabilities. Nevertheless, it is still a work in progress as there are still vulnerabilities and high deployment costs.**

Index Terms—**border gateway protocol, bgpsec, resource public key infrastructure**

1. Introduction

The modern Internet consists of multiple smaller networks, the so-called Autonomous Systems (AS). AS are administered by a single organization and are reachable by an IP prefix. These networks can, for example, be companies, local internet providers, or universities. To identify individual Autonomous Systems, each of them gets assigned a globally unique number. These numbers are administered by the Internet Assigned Numbers Authority (IANA) and assigned to Regional Internet Registries (RIR), who assign them further.

Due to this distributed nature of the Internet, there is a necessity for routers to exchange information about networks they can reach, allowing them to decide where to forward received packets. This exchange of information is called routing. Routing between AS is called external routing, and BGP is the de-facto standard protocol used for this. Over time it evolved to its current 4th version as described in RFC 1105 [1]

While creating a high standard of interconnectivity, BGP lacks in ensuring security. Over time, this leads to some devastating effects globally due to either accidental misconfiguration or malicious intent. Since then, multiple approaches to add security to the protocol were formulated. BGPSec is one of these approaches and the topic of this paper.

The rest of the paper is structured as follows. In Chapter 2, the routing process of BGP is explained. Chapter 3 analyzes the vulnerabilities of the current state of BGP. Afterward, Chapter 4 follows an introduction to BGPSec, focusing on what it tries to achieve. Chapter 5 compares what vulnerabilities BGPSec solves and what attacks are still possible. In chapter 6, this is accompanied by a discussion of the deployment hurdles BGPSec has to overcome to become the new standard. The paper ends with a conclusion on whether the additional security justifies the effort that has to be taken to deploy BGPSec in Chapter 7.

2. The BGP Routing Process

The BGP belongs to the family of path-vector protocols. In path-vector protocols, the most important exchanged routing information is a destination, and path packets have to traverse to reach this destination. Destinations come as an IP prefix, and paths come in the form of a list of AS numbers. The exchanged information is called Network Layer Reachability Information (NLRI).

For two routers to be able to exchange NLRI, they first have to establish a direct connection. This connection is built upon a Transmission Control Protocol (TCP) connection and called BGP peer relationship. To establish this peer relationship, the two peers exchange `OPEN` messages to negotiate parameters of a peer relationship. Such parameters are, for example, capabilities like the use of BGPSec or a maximum time interval the connection will be kept open in case they do not exchange messages. This time interval is called hold-timer and is used to evaluate whether a peer relationship is still active. If the peers do not exchange messages for one full hold-timer, the connection between the two peers is closed. This closing leads to them dismissing all routing information they gained from this connection. To prevent this, peers regularly exchange `KEEPALIVE` messages to reset the hold timer. `UPDATE` messages carry the actual NRLI. The last class of messages specified by BGP is `NOTIFICATION` messages. Peers use these messages to inform other peers about possible errors such as malformed packets.

Figure 1 shows an exemplary routing process. AS1 announces in messages 1) and 2) the prefix 192.0.20./24 to both its peers AS2 and AS3. In the red path, after receiving message 1), AS2 prepends its own AS number to the path and sends message 3) to AS5 with the updated path attribute. In the green path, both AS3 and AS4 prepend their AS number, as one can see in messages 4) and 5). This routing example results in AS5 receiving the two `UPDATE` messages 3) and 5), announcing the same prefix. Because the prefix of both of the messages is identical, AS5 can choose what path to prefer. The router could base the decision on the length of the path leading

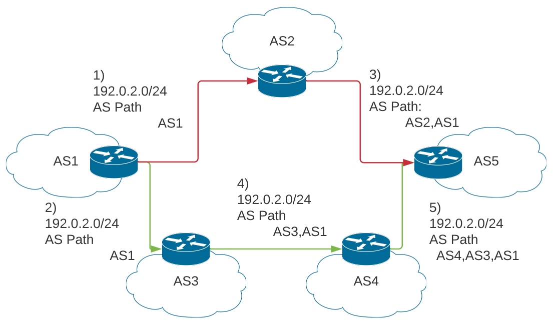


Figure 1: BGP routing process

to AS5 preferring the red path. Another deciding factor could be the economic relationship of the peers. Consider the example of AS2 being a provider and AS4 being a consumer of AS5. AS5 then would have to pay AS2 for traffic but get paid from AS4 for traffic. This monetary difference may lead to a preference for the green path despite it being longer. This decision process is called policy-based routing and can be configured by the router administrator.

3. BGP - Security Concerns

BGP version 4 (RFC 4271) [2], addresses connectivity and scalability demands but makes no considerations towards security. This lack of security makes it easy for accidental or malicious misconfiguration that can devastate the Internet as a whole. As BGP uses TCP as the underlying protocol, TCP's known weaknesses can serve as an additional attack vector. Possible attacks on TCP include attacks such as eavesdropping, insert forged BGP `UPDATE` messages, and Denial of Service attacks. [3]

3.1. Prefix Hijacking

Prefix hijacking is a common attack type in BGP. Its goal is to hijack traffic headed to a specific destination by announcing a more specific prefix to the destination's one. This attack exploits the mechanism of more specific prefix matching.

More specific prefix matching is a standard in routing and used to choose a fitting entry in a routing table in case an IP address matches more than one entry. Consider the example of a router with two entries 1) Destination: 123.4.5.0/24 Next Hop: AS3 and 2) Destination: 123.4.0.0/16 Next Hop: AS5. In the case this router receives a packet with the destination IP 123.4.5.6, it has to choose to what AS it forwards the packet to as both entries fit this IP address. In this case, the router executes more specific prefix matching by preferring the "longest" prefix. In this example, the packet would be forwarded to AS3.

Prefix hijacking makes use of routers' ability to announce arbitrary prefixes and more specific prefix matching. This allows a router to hijack traffic bound to a prefix by announcing a more specific version of it.

Prefix hijacking can be divided into two categories. 1) Black Hole attacks and 2) Interception attacks. The difference between them both is the way they handle the hijacked traffic. In Black Hole attacks, the traffic is attracted and then dropped. Instead of dropping the packets, Interception attacks forward them to the original destination creating a Man in the Middle (MitM) attack enabling the attacker to read and alter packets.

One prominent example of a Black Hole attack is the Pakistan Youtube hijack. This also serves as a good example that even accidental misconfiguration can cause great harm. In 2008, Pakistan made plans to block Youtube country-wide [4]. The Pakistani government instructed Pakistan Telekom to realize this block. They attempted to announce a more specific prefix than the one Youtube announced. and by this, attracting all the traffic originally bound to Youtube. A simplified structure of this attack can be seen in Figure 4.

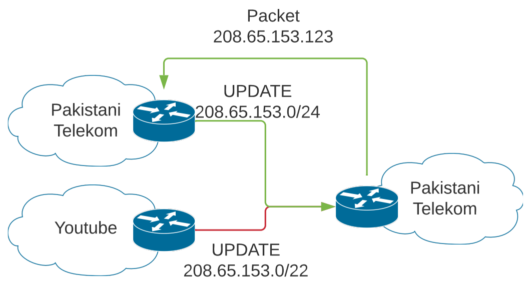


Figure 2: Pakistani Youtube Hijack

Pakistan Telekom announced 208.65.154.0/24. Because of the global propagation of prefixes and /24 being more specific than /22, they got preferred by most existing BPG routers. The green path indicates this. This brought Youtube eventually down for about two hours. The hijack was solved by Youtube announcing even more specific prefixes, effectively hijacking their traffic back.

3.2. Impact on the Internet

As seen in the Youtube hijack incident, a single announcement can greatly impact the Internet. As the Internet traffic steadily grows, so does the amount of sensitive data on the Internet. Recent BGP hijacks show that primary goals were companies that hold vast user data, such as Amazon, Facebook, Google, and Banks. [5] This makes securing BGP a significant concern.

4. BGPSec - an Extension to BGP

Efforts to address BGPs vulnerabilities led to a multitude of different approaches over time. One proposal was to introduce path validation to the protocol. The BGP Security Extension in RFC 8205 [6] formulates this proposal.

4.1. Goals of BGPSec

The introduction of path validation and origin validation intends to harden BGP to achieve byzantine robustness. Byzantine robustness is described as in case of malicious or faulty behavior of hosts, the other hosts should 1) receive the same message that was sent by the original host 2) reach a decision on a message's contents within a finite time period 3) this decision should be the same among all these hosts [3].

4.2. Path and Origin Validation

Path validation is a mechanism that allows routers to validate the path information contained in `UPDATE` messages. The validation checks whether the announced path matches the actual path packets will take. Origin validation asserts whether the announcing AS owns the prefix contained in the `UPDATE` message. For routers to execute these validations, an additional infrastructure is needed that holds information about AS numbers and prefix owners. A possible implementation of such an infrastructure is called Ressource Public Key Infrastructure.

4.3. Resource Public Key Infrastructures

The Resource Public Key Infrastructure (RPKI) is used to issue and distribute certificates that link resources to resource holders. Such resources can be IP prefixes and AS numbers. [7]. These certificates are then published and made available for the public on dedicated repository servers. These certificates can be queried for matching AS numbers, IP prefixes, and Subject Key Identifiers (SKI). These are identifiers used in case an AS number corresponds to multiple certificates.

4.4. Certificate Issuing Process

The certificate issuing process is hierarchical and in accordance with the allocation of IP address space. Consider the following example: IANA allocates the address space 123.0.2.0/24 and the AS number 20 to the Regional Internet Registry RIPE NICC. RIPE NICC, in turn, allocates the AS number and address space to a university network.

IANA then would assign a certificate to RIPE NICC holding the authority to use the AS number 20 and a certificate holding the authority to announce IP prefixes in 123.0.2.0/24. As these certificates authorize an entity to announce particular prefixes, they are called Route Origin Authentication (ROA). RIPE NICC subsequently issues another set of certificates to the university and publishes the certificates in a publicly accessible repository server.

4.5. Exemplary BGPSec Routing Process

In BGPSec, the `AS PATH` attribute gets replaced with the `BGPSec PATH` attribute to hold the additional information in the form of a signature block. Each signature in that block corresponds to an AS number in the path. So the longer the path gets, the more signatures such a block will contain. The way signatures are created is based on whether the router announces a prefix or propagates routing information. A prefix-announcing router create the signature based on the announced prefix, their own AS number, and the AS number they forward the `UPDATE` message to. On the other hand, a router that propagates a received `UPDATE` message uses the previous signature instead of a prefix to create a new signature. Each of these signatures is accompanied by a SKI.

Figure 3 shows an exemplary routing process. AS1 announces its prefix 192.0.2.0/24. It begins by prepending its AS number to the `Secure Path` and then create a signature block corresponding to its AS number. When AS2 receives the `UPDATE` message sent by AS1, it validates all signatures in the signature block and then appends its own AS number to the `BGPSec Path` and a new signature to the signature block. After this, it forwards the `UPDATE` message to AS3. The router at AS3 then again validates the information. As now two signatures are contained in the signature block, the router at AS3 has to validate two signatures. It begins with the most recent one, in our example sig2. To validate it, it queries the certificate that matches SKI2 and AS2. If a matching certificate is found, they use the public key to validate the signature cryptographically. If this validation fails or no certificate was found, the `UPDATE` message will be deemed invalid. Then, the router validates the next signature the same way. After validating the last signature, the router can query the ROA corresponding to the contained prefix.

Based on the validated signatures, the router can ensure that each AS number in the path belongs to the router that created the signature. Furthermore, the router can ensure that the next hop in the signature corresponds to the next AS number in the path. By this, path validation is achieved. Moreover, by querying the ROA, the router can ensure that the origin AS is allowed to announce the prefix. With this, Origin Validation is ensured.

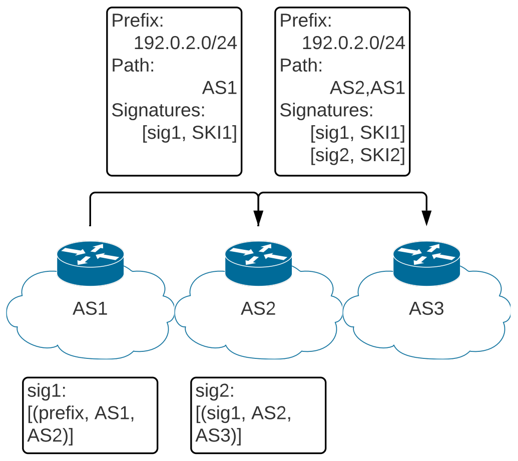


Figure 3: BGPSec Routing Process

5. Analysis of the Effectiveness of BGBPsec

As mentioned above, BGPSec only ensures valid origins and that paths are genuine. While preventing attacks to some degree, there are still vulnerabilities that have to be addressed.

5.1. What It Prevents

BGPSec covers most accidental misconfiguration and unsophisticated attacks. For example, the Youtube hijack from chapter 3 would be prevented. The first BGP router that receives the `UPDATE` message would try to validate the signature. This validation attempt would fail because Pakistani Telekom did not use Youtube's private key as it is, at least in theory, not in their possession.

5.2. What It Does Not Prevent

One major issue with BGPSec as it is at the moment is, that traffic hijacking is still possible. A good example is the wormhole attack. The basic idea of that attack is to create a shorter path than the current one and by this redirecting the traffic. Figure 4 shows such an attack.

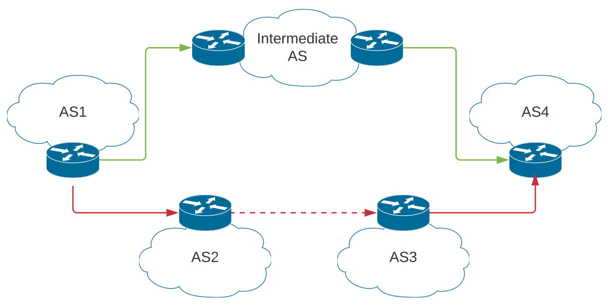


Figure 4: Wormhole Attack

To conduct a wormhole attack, an attacker needs to control two BGP speakers´. These have to be in a peer relationship with the endpoints of the traffic the attacker wants to hijack. In this example, the attacker is in control of speakers at AS2 and AS4. Before the attack, the traffic between AS1 and AS4 is forwarded via the green path. The attacker now creates a tunneled peer relationship between AS2 and AS3 indicated by the red dashed line. This creates a path of length three between AS1 and AS4. In about 86 percent of cases [8], path lengths are longer than 3. This leads to the red path being preferred due to shorter path length and leads to a MitM attack. Because the attacker does not announce a prefix and does not forge the `BGPSec PATH` attribute, this attack is not covered by BGPSec.

6. Discussion of Deployment Issues

Additional security comes with an additional cost in terms of storage and processing power requirements. This cost and the still obvious flaws BGPSec has played a major role in the hurdles it has to overcome in terms of deployment.

6.1. A Technical View

The introduction of signatures and certificates to the routing process is crucial for the additional security BG-PSec offers. Each time a BGP router receives an `UPDATE` message, it validates all signatures. Additionally, each time a router propagates an `UPDATE` message, it has to create a signature and append it to the signature block in the `BGPSec Path`. This leads to an increasing number of validations the more extended the path gets, and makes it on average 70 times slower than regular BGP [8]. The number of required `UPDATE` message itself also increases. This is because `BGPSec Path` attributes can only contain one single prefix while regular `AS PATH` attributes can contain multiple ones [6]

This serves as a significant hurdle because, according to RFC 7747 [7], convergence is a major factor in the reliability of BGP. Convergence means that all routers have the same information about the network topology. Due to the continuous change of this topology, the propagation of this change should happen fast. Because of BGPSec's longer processing time, convergence is slower than the convergence without it.

The upside of BGPSec is that as an extension, it can work in parallel with regular BGP. As stated in the RFC, the BGPSec Path attribute is an optional attribute that replaces the AS path attribute. The decision of what attribute to use is negotiated between peers using the `OPEN` messages. In the case that a BGP speaker wants to propagate a prefix is received from a peer connected using BGPSec, the BGPSec path attribute will get stripped of its additional information and then propagated as AS path to the peers that do not use BGPSec. This allows for a gradual deployment because BGP and BGPSec routers can coexist, and communication between them does not affect the routing process. in the scope of insecure routing.

6.2. A Management View

As stated in RFC 4271 [2], the management of certificates and origin/prefix pairs are handled by two distinct RPKIs. Because BGP is not under a single authority, collecting complete data sets and keeping them up to date is a significant deployment hurdle BGPSec still has to face. At the moment, there is already an RPKI in place for origin authentication. According to a report on RPKI [9], about 27 percent of prefix announcements are valid, 0.5 invalids, and 72,4 unknown, meaning that the RPKI has no information about the pairing of prefixes to AS numbers. This shows that there were some efforts to implement it, but wide-scale deployment has yet to be achieved. Additionally, there is no incentive to provide such data for a single ISP because they get no direct value out of this. An approach to change this may be the deployment beginning with more prominent parts of the Internet and discrimination of the ones that did not implement it by preferring connections through paths using BGPSec.

7. Conclusion

This paper presented BGP as the de-facto standard routing process to exchange routing information between AS. As BGP has no built-in security, it is vulnerable to attacks such as the famous Pakistan Youtube incident. BGPSec is a proposed extension to BGP that adds path and origin validation to the routing process by using RPKI. However, although it prevents some attacks and misconfiguration from happening, there are still significant flaws that allow for attacks like wormhole attacks. These vulnerabilities show the state of the protocol as a work in progress. Contributing to that are the still prevalent issues it faces in terms of deployment.

The additional security BGPSec offers comes with a price. The creation of signatures and the validation process takes more time and needs additional space. Furthermore, it is hard to collect and manage the necessary data, as there is no single authority that manages AS numbers and prefixes. With BGP being an old protocol, these problems follow the problems other protocols of that era face, like DNS. With approaches to improve BGPSec existing but not yet included in the current RFC, more research is still necessary to find a fitting solution to the current problems BGP is facing.

References

[1] K. Lougheed and J. Rekhter, "Border gateway protocol (bgp)," Internet Requests for Comments, RFC Editor, RFC 1105, June 1989, http://www.rfc-editor.org/rfc/rfc1105.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1105.txt

[2] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Internet Requests for Comments, RFC Editor, RFC 4271, January 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4271.txt

[3] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of bgp security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[4] (2008) Youtube hijacking: A ripe ncc ris case study. [Online]. Available: https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study

[5] J. Sherman. (2020) The politics of internet security: Private industry and the future of the web. [Online]. Available: https://www.atlanticcouncil.org/in-depth-research-reports/report/the-politics-of-internet-security-private-industry-and-the-future-of-the-web/

[6] M. Lepinski and K. Sriram, "BGPsec Protocol Specification," Internet Requests for Comments, RFC Editor, RFC 8205, September 2017. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8205.txt

[7] M. Lepinski and S. Kent, "An infrastructure to support secure internet routing," Internet Requests for Comments, RFC Editor, RFC 6480, February 2012, http://www.rfc-editor.org/rfc/rfc6480.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6480.txt

[8] K. Kim and Y. Kim, "Comparative analysis on the signature algorithms to validate as paths in bgpsec," in *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, 2015, pp. 53–58.

[9] (2020) Global prefix/origin validation using rpki. [Online]. Available: https://rpki-monitor.antd.nist.gov/

IEEE 802.1Qcr Asynchronous Traffic Shaping with Linux Traffic Control

Christopher Pfefferle, Florian Wiedner*, Christoph Schwarzenberg*
**Chair of Network Architectures and Services, Department of Informatics*
Technical University of Munich, Germany
Email: ga38pav@mytum.de, wiedner@net.in.tum.de, schwarzenberg@net.in.tum.de

Abstract—**The widespread TSN standards, as introduced by the IEEE 802.1 working group, provide low latency scheduling and shaping with guaranteed packet transfers. Until recently, they depend on a synchronized clock between all network nodes. The newly introduced ATS standard however revokes this dependency for software with real-time requirements and can introduce TSN to a wider community. This paper describes the requirements to implement the ATS standard using the Linux TC tool for traffic shaping and scheduling. While it is currently impossible to implement LRQ queues or the Paternoster scheduler with TC commands, a model of the UBS algorithm with TBE queues will be presented.**

Index Terms—**asynchronous traffic shaping, time sensitive network, traffic control, traffic scheduling**

1. Introduction

Network standards take ever-growing steps to revolutionize our daily lives. Presentations are held online with participants scattered all around the globe and even presentation systems in offices send slides from the presenter over the local area network to the monitor. In those scenarios a slight delay will not irritate the speaker, but when dealing with precision machinery a delay of only milliseconds can break fragile components. For these use cases the IEEE 802.1 working group creates standards for Time-Sensitive Networking (TSN) to provide communication protocols and features that can deliver precise communication.

A major contribution was the introduction of synchronization between participating devices in a network, introduced by the IEEE 802.1BA Audio Video Bridging (AVB) standard, allowing precise traffic scheduling and guaranteed packet deliveries [1]. However, the time synchronization mechanism adds high complexity to network setup and maintenance. The IEEE 802.1Qcr Asynchronous Traffic Shaping (ATS) standard intends to bypass the complexity of synchronization by revoking it and allowing every network node to send traffic on its own timing [2]. Nevertheless, it still aims to achieve deterministic and low transmission delays.

In order to shape and schedule traffic on Linux machines a program is required to intervene in the path of packets from creation to the network interface. One such tool is traffic control (TC), it comes pre-installed on Linux distributions and provides powerful possibilities to control the network traffic of a computer [3]. With ATS relaxing the requirements for TSN and TC moreover providing the necessary tools, daily used software is able to establish real-time requirements and further facilitate in the ever-growing impact of the internet.

This paper gives an overview over ATS and Linux TC, presents a possible model to implement a part of the standard with the tool and points out its current limitations. It is structured as follows: Section 2 will give a short history of related work while Section 3 describes implementation details of ATS and a selective overview over the possibilities of TC. In Section 4 the details of how TC can be utilized to implement the requirements of ATS will be shown, and Section 5 will conclude this work and suggest next steps.

2. Related work

The first part of this Section will provide a brief overview over the development of TSN and the background of the ATS standard and its latest developments. The second part will introduce the Linux TC command and its contributions in networking on Linux machines, as well as its relevance.

2.1. Asynchronous Traffic Shaping

Asynchrony in network traffic has been established as the standard in shared networks for a long time and is the backbone of the Internet connecting millions of nodes. Rigolio et al. proposed shaping on the Asynchronous Transfer Mode (ATM) as early as 1991, offering control over the bandwidth and flow exiting a given system [4].

In 2011 the IEEE 802.1BA standard for AVB was approved, proposing the features for TSN that are still relevant today [1]. It introduced time synchronization between network devices and traffic shaping, to minimize delay and jitter for distributed systems with real-time constraints.

Ahead of the introduction as an IEEE standard, in 2016 Specht and Samii introduced the Urgency-Based Scheduler (UBS) in [5] with two proposed algorithms: Length-Rate Quotient (LRQ) and Token Bucket Emulation (TBE). They are both based on Rate-Controlled Service Disciplines (RCSDs) and therefore can provide guarantees on both deterministic and statistical performance by separating the scheduling and shaping components [6].

One year later UBS and the then newly introduced Paternoster scheduler, which is based on Cyclic Queuing and Forwarding (CQF) [7], were approved as an official standard by the IEEE TSN working group within the

IEEE P802.1Qcr ATS amendment. Zhou et al. have compiled a detailed insight into ATS together with a performance evaluation in various simulated environments [8].

Because of its recent publication there are not many works incorporating it yet; performance evaluations are given by Specht and Samii in [9] and by Zhou et al. in [10], and Mohammadpour et al. provides computed worst-case bounds for latency and backlog in [11]. Le Boudec analyzes a more generalized approach of UBS in [12] and Grigorjew et al. propose an addition to increase jitter control in [13]. Also new introduced standards take ATS into account, the IEEE P802.1Qdd Resource Allocation Protocol standard incorporates support for ATS [14].

The most recent performance assessment of ATS was released by Fang et al. in November 2020, taking various released standards of the TSN working group into account, revealing performance advantages of ATS especially in heavy-load cases [15].

2.2. Linux traffic control

The Linux TC tool was introduced in 2001 with the Linux kernel version 2.2, within the iproute2 package. An influential work is done by Hubert, combining definitions and application examples in a well-arranged document and continuing updated support on his online HOWTO document [16].

TC is a powerful tool for distributing and shaping network traffic, allowing the user to define detailed rules. It is used to rule over multiple services communicating through a network with restricted capabilities, examples like the work done by Vila-Carbo et al. in [17] show that its qualities are capable to define rules for real-time transmissions.

The introduction of an implementation of the Credit-Based Shaper (CBS) for TC [18], as defined by the IEEE 802.1Q-2014 standard, further shows the potential of TC for TSN applications and is used to implement synchronous traffic shaping on computers using Linux-based operating systems.

3. Architecture Details

Here, a short selective overview over the architecture of the ATS standard and Linux TC will be given. Particularly the later is a powerful tool to work with and has an accordingly large documentation. The focus will therefore lay on a subset of the traffic shaping capabilities needed to compare the possibilities of TC with the requirements of ATS in Section 4.

3.1. ATS algorithms

This Section is mainly based on [8]. The idea behind ATS is the independent clock of every connected device in a network, discarding the problems that arise when distributed devices have to agree on a synchronized timer. Its main requirement are queues that support asynchrony. A shaper is bond to each which assigns eligibility times to the frames in the queue, and on this information a transmission selection algorithm decides when frames are transmitted. This algorithm can be described as a simple gate control, taking the eligibility times into consideration. These initial shaped queues are simple FIFO queues and they ensure the processing of high-priority flows is not affected by malicious or other interfering flows.

The shaped queues need to follow the queue allocation rules, direct quoted from [8]:

QAR1:
frames from different transmitters are not allowed to be stored in the same shaped queue.

QAR2:
frames from the same transmitter but not belong to the same priority in the transmitter are not allowed to be stored in the same shaped queue.

QAR3:
frames from the same transmitter with the same priority in the transmitter, but not belong to the same priority in the receiver are not allowed to be stored in the same shaped queue.

After shaping the eligible frames, they are sent from the shaped queues and stored in shared queues. These are managed by one of the shapers described below which selects and forwards them to the network interface releasing them into the network.

The ATS standard proposes two scheduling algorithms which can be used to realize asynchronous shaping queues, the foremost introduced UBS algorithm and the Paternoster algorithm.

The UBS scheme allows two types of shaped queues to be used: LRQ and TBE, respectively based on the frame-by-frame leaky bucket algorithm and token-based leaky bucket algorithm [6].

LRQ disregards the incoming flow pattern and shapes with a stabilized transmitting/leaking rate, by calculating the eligibility time of a packet as "the quotient between the size of the previously transmitted frame and the reserved link rate of the particular class" [8].

TBE allows some level of bursty traffic transmitting while maintaining an average rate, it uses the accumulation time of "tokens" in a "bucket" to calculate the eligibility time of a packet. In comparison to LRQ, it provides a better utilization of the given bandwidth on a lighter load.

The scheduling is achieved using the ATS algorithm based on a Leaky Bucket approach. Frames are processed with respect to their eligibility times, their arrival time, the size of the last frame, and the current system clock allowing to drop overdue frames. Therefore, the ATS scheduler acts as the final shaper for the available bandwidths.

The Paternoster queuing and scheduling algorithm is a cyclic approach. It utilizes four queues, which in every epoch pass through one of the four states *prior*, *current*, *next* and *last* as depicted in Table 1. In each epoch, frames are enqueued into the *current* queue, if it is full they are passed through to the *next* or *last* queue or get dropped, and are only dequeued from the *current* queue. The epoch length will influence the delay and has to stay consistent within the network. [8]

TABLE 1: Queuing in Paternoster, adapted from [8]

Queue / Epoch	Queue0	Queue1	Queue2	Queue3
Epoch 0	prior	**current**	next	last
Epoch 1	last	prior	**current**	next
Epoch 2	next	last	prior	**current**
Epoch 3	**current**	next	last	prior
…				

3.2. TC for traffic shaping

This Section is largely based on the information compiled by [16]. Every network packet, which can either be produced by a local program or is being forwarded, has to pass through the TC architecture.

The main components are Queuing Disciplines (qdiscs), i.e., specified queuing algorithms. They can be either classless or classful, the later supporting an internal division into classes that again contain configurable qdiscs. Hence, they are arranged in a tree structure, with the Linux kernel interacting (enqueuing and dequeuing of network packets) with the root node only. A qdisc can perform three actions on packets queued into it: (1) scheduling, i.e., prioritization of packets over others, (2) shaping, i.e., delaying or even dropping packets to satisfy maximum traffic rate requirements, and (3) policing (if used on incoming traffic), i.e., dropping packets to satisfy internal requirements on incoming traffic.

If a qdisc may delay packets for the purpose of maintaining a constant transmission rate it is considered to be non-work-conserving. If it, on the other hand, sends out packets as soon as they are available, it is considered work-conserving.

Finally, a filter can be assigned to each class, it holds conditions with which packets can be classified. If a filter matches a packet, it will be forwarded to the qdisc of the respective class. In the following the qdiscs addressed in this paper are explained:

pfifo_fast
The default qdisc, a classless shaper with three FIFO queues, one for each priority level as defined by the Type of Service (TOS) flag of network packets. Packets are dequeued starting from the highest priority queue.

Token Bucket Filter (TBF)
A classless shaper that supports a set maximum rate with short bursts.

Stochastic Fairness Queuing (SFQ)
A classless scheduler that dequeues packets in a round-robin fashion through flows, which mostly correspond to a TCP/IP connection.

PRIO
A classful scheduler similar to pfifo_fast, but it supports enqueuing with filters and subclasses other than simple FIFO queues. Dequeuing is done the same way, starting at the defined first qdisc.

Hierarchical Token Bucket (HTB)
A classful shaper that allows to precisely limit the bandwidth of its child qdiscs with a possibility to borrow unused resources.

Clark-Shenker-Zhang algorithm (CSZ)
A complicated classful scheduler proposed by the three eponymous researchers in [19], providing guaranteed service for real-time applications along with best-effort queues, reducing delay and jitter by deliberately not shaping.

Credit-Based Shaper (CBS)
A classless shaper that implements the CBS algorithm as introduced in the IEEE 802.1Qav standard, relying on set bandwidths.

Earliest TxTime First (ETF)
A classless shaper that is constructed to support shaping in TSN, dequeuing packages on a configurable timer.

4. Implementation of ATS with TC

With the requirements and available tools introduced, a possible implementation of the ATS standard with the UBS scheme and TBE queues is presented. Section 4.2 will then explain the problems encountered when trying to implement LRQ queues or the Paternoster scheme. Both Subsections refer to details of the ATS algorithm as requirements and the functionalities of TC as capabilities to implement the former. For detailed descriptions and definitions refer to Section 3 and the corresponding Subsections.

4.1. A TC model of UBS/TBE

The root qdisc of the TC scheme does not necessarily need any shaping capabilities, shaping will be achieved by further qdiscs. But to contain further classes it needs to be classful. Its purpose is the separation of packets into queues with regards to their priorities, so packets with the same priority are handled by the same queue. This enforces the separation of the different priorities as required by QAR2. As the distribution of packets into classes can be controlled by filters, the focus lies on the correct dequeue strategy. The root qdisc must abide to the ATS scheduling algorithm. Potential candidates as root nodes include a PRIO qdisc that achieves a strict prioritization, the CSZ algorithm that provides guaranteed service for high-prioritization packets while possibly neglecting lower-prioritization ones, and HTB which is a generally good competitor that provides no strict prioritization and acts as a shaper, which may introduce delay and jitter. The ATS algorithm on one hand performs shaping, which would render the HTB as the best option, but on the other hand it also performs strict prioritization which would require a PRIO qdisc. As a solution, both will be used to account for the requirements of the ATS algorithm.

Each class assigned to a priority level has to implement the requirements of the TBE queue. A viable option is the TBF shaper applying nearly the same token/bucket technique, but as it is a classless qdisc its application at this point would violate QAR1.

The only qdisc realizing both QAR1 and QAR3 is the classless SFQ scheduler which permits to split the packages by conversations. As it does not support shaping, a separate shaper is needed.

```
                 +----------------+
                 | root qdisc HTB |
                 +----------------+
                         |
                    +---------+
                    | class 1 |
                    +---------+
                         |
                   +------------+
                   | qdisc PRIO |
                   +------------+
                 /       |       \
+-------------+ +-------------+ +-------------+
| class prio1 | | class prio2 | | class prio3 |
+-------------+ +-------------+ +-------------+
       |               |               |
 +-----------+   +-----------+   +-----------+
 | qdisc SFQ |   | qdisc SFQ |   | qdisc SFQ |
 +-----------+   +-----------+   +-----------+
```

Figure 1: Proposed TC scheme for ATS

The solution that is presented in this paper is the usage of a HTB root node with only one subclass, a PRIO qdisc. With this setup, as shown by Figure 1, a SFQ qdisc can be inserted per priority level to ensure all three queue allocation rules are met.

4.2. Limitations

LRQ queues cannot be realized with the qdiscs provided by TC, because it heavily relies on the calculation of eligibility times and their propagation through the scheme. ETF is currently the only qdisc to allow the calculation of eligibility times. Because it is classless, it lacks the requirements to be used as a root node by itself, and as the calculated times cannot be used in further schedulers a new qdisc would be required to account for the scheduling done by the ATS algorithm considering the timer of ETF.

Similarly, Paternoster is not realizable with the available tools. It requires en- and dequeuing into and from different queues depending on the current epoch, which can neither be accomplished with the current qdiscs nor with filters.

5. Conclusion and future work

The IEEE 802.1Qcr ATS standard was introduced to increase the real time capabilities of networks that have no synchronized timer available for every node, reducing the requirements while still providing guarantees for time sensitive applications. Using the TC tool this paper shows a possible model to realize this standard, in particular the UBS scheme with TBE queues, on Linux machines. Using only the HTB, PRIO, and SFQ qdiscs it is possible to comply to the conditions of ATS.

It further highlights the problems that arise when working on the other possible schemes of the standard, namely UBS with LRQ queues and the Paternoster queuing/scheduling algorithm. With most of the requirements met, TC currently lacks key features like the usage of eligibility times or dynamically changing the roles of queues. However, it seems possible to add these features in future updates.

The next step would be an implementation of the proposed scheme to analyze its practicability and performance. Even though TSN standards are only applied in a specialized field, if this approach turns out feasible it may allow more general applications in Smart Homes or the Internet of Things.

References

[1] "IEEE 802.1BA-2011 - IEEE Standard for Local and Metropolitan Area Networks--Audio Video Bridging (AVB) Systems," https://standards.ieee.org/standard/802_1BA-2011.html, 2011, [Online; accessed 25-March-2021].

[2] "P802.1Qcr – Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping," https://1.ieee802.org/tsn/802-1qcr/, 2018, [Online; accessed 20-March-2021].

[3] B. Hubert, "iproute2 - TC (8)," Linux man page (8), 2001.

[4] G. Rigolio, L. Verri, and L. Fratta, "Source Control and Shaping in ATM Networks," in *IEEE Global Telecommunications Conference GLOBECOM '91: Countdown to the New Millennium. Conference Record*, vol. 1, 1991, pp. 276–280.

[5] J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 75–85.

[6] H. Zhang and D. Ferrari, "Rate-Controlled Service Disciplines," *Journal of High Speed Networks*, vol. 3, no. 4, pp. 389–412, 1994.

[7] "P802.1Qch – Cyclic Queuing and Forwarding," https://1.ieee802.org/tsn/802-1qch/, 2016, [Online; accessed 25-March-2021].

[8] Z. Zhou, M. S. Berger, S. R. Ruepp, and Y. Yan, "Insight into the IEEE 802.1 Qcr Asynchronous Traffic Shaping in Time Sensitive Network," *Advances in Science, Technology and Engineering Systems Journal*, vol. 4, no. 1, pp. 292–301, 2019.

[9] J. Specht and S. Samii, "Synthesis of Queue and Priority Assignment for Asynchronous Traffic Shaping in Switched Ethernet," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 178–187.

[10] Z. Zhou, Y. Yan, M. Berger, and S. Ruepp, "Analysis and Modeling of Asynchronous Traffic Shaping in Time Sensitive Networks," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, pp. 1–4.

[11] E. Mohammadpour, E. Stai, M. Mohiuddin, and J.-Y. Le Boudec, "Latency and Backlog Bounds in Time-Sensitive Networking with Credit Based Shapers and Asynchronous Traffic Shaping," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 02, 2018, pp. 1–6.

[12] J.-Y. Le Boudec, "A Theory of Traffic Regulators for Deterministic Networks with Application to Interleaved Regulators," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2721–2733, 2018.

[13] A. Grigorjew, F. Metzger, T. Hossfeld, J. Specht, F.-J. Götz, F. Chen, and J. Schmitt, "Asynchronous Traffic Shaping with Jitter Control," 2020.

[14] "P802.1Qdd – Resource Allocation Protocol," https://1.ieee802.org/tsn/802-1qdd/, 2018, [Online; accessed 20-March-2021].

[15] B. Fang, Q. Li, Z. Gong, and H. Xiong, "Simulative Assessments of Credit-Based Shaping and Asynchronous Traffic Shaping in Time-Sensitive Networking," in *2020 12th International Conference on Advanced Infocomm Technology (ICAIT)*, 2020, pp. 111–118.

[16] B. Hubert, "Linux Advanced Routing & Traffic Control," in *Proceedings of the Ottawa Linux Symposium*, 2002, pp. 213–222.

[17] J. Vila-Carbo, J. Tur-Masanet, and E. Hernandez-Orallo, "An Evaluation of Switched Ethernet and Linux Traffic Control for Real-Time Transmission," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 400–407.

[18] V. C. Gomes, "tc-cbs (8)," Linux man page (8), 2017.

[19] D. D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," ser. SIGCOMM '92. Association for Computing Machinery, 1992, pp. 14–26.

Certificate Revocation

Raphael Schmid, Juliane Aulbach*, Patrick Sattler*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: raphael.schmid@tum.de aulbach@net.in.tum.de, sattler@net.in.tum.de

Abstract—
This paper explains the concept of certificate revocation and how it is implemented and enforced in the real world. It explains why certificate revocation is necessary and how there is no agreed-on standard to execute it. It will introduce the actors in certificate revocation and how they fulfill their role in ensuring user security. Then the paper will explain various approaches to optimize certificate revocation and compare them to conclude which one is the most suitable for usage.

Index Terms—**certificate revocation, internet security, revocation methods**

1. Introduction

In the TLS protocol, having the ability to revoke certificates when they become invalid before their validity period ends is crucial. However, there is no agreed-upon standard in the industry as to how to handle revocation. It results in most browser clients not adequately checking for revocation when connecting with a server. It can allow attacks like "man-in-the-middle attacks". This paper introduces the necessary background information to understand certificate revocation in chapter 2. Then it explains how certificate revocation works and its most used methods in chapter 3. In chapter 4, the behavior of browsers is explored and what their different shortcomings are when it comes to certificate revocation. Then in Chapter 5, it discusses which problems can arise from the server, which must request a certificate revocation if a certificate gets compromised. Chapter 6 explores which different classifications there are for handling certificate revocation and then introduces some of the most recent approaches, concluding with a comparison of said approaches. Chapter 7 then evaluates the findings of the previous chapters and Chapter 8 finishes with a conclusion of the paper.

2. Background

The secure exchange of data on the internet is a big concern for many people nowadays. A widely used standard to ensure this security is the TLS protocol. TLS stands for Transport Layer Security. It utilizes a twofold system. One part of this system is encryption to ensure confidentiality and integrity of the data. The other part is to verify the identity of a remote party using digital certificates. [1], [2]
These digital certificates will be the main focus of this paper. Here is a real-world example of how they work. Say a client, e.g. a web browser, wants to connect to a server. To verify its identity to the client, the server sends a digital certificate to the client, which is specifically issued for the domain of the server. This digital certificate identifies the server to the client. But with just that, neither the validity of the server nor the certificate is guaranteed. This is attained using a so-called *chain of trust*. [1], [2]
The chain starts with a certificate authority (CA) which presents a *root certificate*. A CA can issue a certificate for any domain. The issued certificate gets signed by the CA ensuring that the certificate can always be traced back to the CA. If this certificate now can also issue and sign certificates itself, it is called an *intermediate certificate*. A certificate that is not able to sign another certificate is called a *leaf certificate*. Leaf certificates are used by most websites. *Root certificates* are assumed to be trusted by the client. When a client now wants to verify a certificate it has to follow this chain, starting with the root certificate, traversing zero or more intermediate certificates until it reaches the leaf certificate. The client has to verify every signature of every certificate it traverses. [1], [2] This mechanism is visualized in figure 1 below.

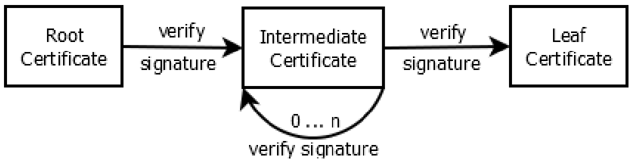


Figure 1: Chain Of Trust

3. Certificate Revocation

Certificates are not valid indefinitely. Every certificate has a validity period [1]. But, there can be occasions where the certificate is not valid anymore before the said validity period is over. Examples for this could be that the private key of a certificate became compromised, was generated with a weak algorithm, or the erroneous issuance of a certificate. [3] If that happens, the certificate must get revoked, advertising to entities checking the certificate that it is no longer valid [1]. Revoking a certificate, in this case, is important. If a certificate got compromised but not revoked, it could open some serious vulnerabilities. For example, an attacker impersonating the identity of the webserver or eavesdropping on private communication. [2]

There are three key actors in certificate revocation:

- Web server / Server administrator [1], [2]
- Certificate authority (CA) [1], [2]
- Client (e.g. web browser) [1], [2]

Here is how a certificate revocation would play out in real life: When a certificate needs to get revoked, first, it is the responsibility of the web administrator to send a revocation request to the CA which signed the certificate. The CA then must sign a statement that the certificate got revoked and is responsible for disseminating this information on the internet. [1], [2] A client that wants to connect to the server managed by the web administrator then is responsible for checking the status of a certificate used by a server to establish a secure connection. There are two methods most often used by CAs to disseminate revocation statuses of certificates online. CRLs and OCSP. [1]

3.1. CRL

A Certificate Revocation List (CRL) is a file that contains a list of all revoked certificates of a CA. Every CRL has an expiration date and must get updated and published periodically by the CA. To check if a certificate got revoked, the client must download the CRL from the CA and then check for the certificate inside the downloaded CRL. The client can cache the CRL until its validity expires. But still, making the client download a whole list of revoked certificates imposes a burden on the client and can add to latency when loading a web page. It is especially true for bigger CRLs. Apple, for example, had a CRL with 2.6 million revoked certificates which had 76 MB in size. [1]

3.2. OCSP

OCSP, Online Certificate Status Protocol, aims to address this problem of high overhead. It reduces the overhead compared to the CRLs by making the client query for the revocation status of just a single certificate. Using OCSP, the client generates an HTTP request to check the serial number of a certificate. The CA then sends the information to the client with a signed response. But this approach introduces a new problem. It contains a privacy issue, exposing the client's browsing behavior to the CA. Also, it still puts some burden on the client and adds to latency. [1]

4. Browsers

In a real-world scenario, the client connecting to a server is typically a browser. This section will explain the behavior and shortcomings of browsers in certificate revocation.
Browsers behave differently on varying devices. There is no mobile browser that checks for certificate revocation. The reason for this lack of security is to decrease the cost, regarding latency and power, for the mobile devices. [1] But, like already discussed, not checking for the revocation status of certificates opens up some serious security vulnerabilities, which is alarming considering that 55.56% of web traffic is generated by mobile phones [4].
When observing the behavior of desktop browsers, there is no common behavior as to how to handle certificate revocation. No browser acts the same. It is also worth noting that even for the same browser, the behavior can change depending on the platform and the type of certificate. Since it is exceeding the scope of this paper to consider all these different factors, this paper will only look at Google Chrome and Mozilla Firefox as examples of shortcomings of browsers handling certificate revocation. Also, only the behavior on Microsoft Windows will be covered, since it is the most used operating system. [5] ([1] takes an in-depth analysis on this matter.)
Google Chrome does not check CRLs for certificate information. Chrome uses a customized CRL to look up revocations called the *CRLSet*. The CRLSet is a 250 kB big, pre-populated list of certificates that get put together by Google by crawling a pre-set list of CRLs. It is not clear what Google's criteria are for these CRLs. This list of CRLs is already just a small subset of the CRLs existing on the internet but Google then also applies a list of rules to drop more revocation data. Which results in Google covering only 75.6% of the revoked certificates that they consider. All of this leads to Google covering only 0.35% of all revoked certificates online in their CRLSet. [1] [2]
Mozilla Firefox uses NSS for certificate verification. "Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled client and server applications." [6] Firefox disabled CRL checking and only uses OCSP requests. Extended Validation (EV) certificates are special certificates containing additional information offering a more careful verification. Firefox differentiates between EV and non-EV certificates, checking only the leaf-certificates for non-EV certificates. This is a behavior displayed by many browsers. [1] Like Googles' CRLSet, Firefox uses OneCRL, a list of pre-stored certificates [7], [8].

5. Servers

Server administrators are responsible for issuing revocation requests to their respective CA. A human component in the process of certificate revocation opens some security vulnerabilities. This showed in the aftermath of the Heartbleed event in 2014, a "vulnerability in OpenSSL's implementation of the Heartbeat Extension" which caused a mass revocation of digital certificates. [9] It exposed that even after 3 weeks 10% of vulnerable websites still had not addressed the issue. And also showed the number of revocations went down on the weekend, probably because there was no server administrator monitoring the security of the server during that time. And finally, of the certificates that were reissued because of Heartbleed 4.1% of certificates were reissued with the same private key. [9]
This raises the question of whether an important task like revoking certificates, should be done manually and be prone to human error.

6. Methods of revocation validation

There are three different types of approaches to revocation validation.
When utilizing a **pull-based** approach, the client requests the revocation status of a certificate when needed [10].

CRLs and OCSP classify as pull-based approaches.
Second is the **push-based** approach, where the client periodically downloads revocation information. This approach does not reveal the client's traffic patterns, while most of the pull-based approaches do. [8]
Last is the **network-assisted** approach. The idea is to change the ecosystem of TLS in a way that makes it unnecessary for the client to request the revocation status of a certificate. [1], [8], [10]

6.1. CRLite

CRLite is a push-based approach. The idea is to push all certificate revocations to the browser periodically utilizing a two-part system. [11] The first part happens on a server, where all known TLS certificates are getting crawled on the web. All of the revoked certificates get hashed into a bit-vector. If the bit assigned to a certificate is 1, the certificate got revoked. To avoid the risk of false positives, confusing a non-revoked certificate with a revoked one, this mechanism gets repeated with a set of all revoked certificates and all non-revoked certificates until no false positives are left. The filter used is called a bloom filter and this technique of cascading down many filters to eliminate false positives is a bloom filter cascade. Avoiding false positives is crucial to allow the client to hard-fail. [11] The second part is on the client-side, downloading the filters and using them to check for the revocation of observed certificates. [11] CRLite aims to maximize the efficiency of checking for revoked certificates. Only 10 MB are needed for roughly 30 million certificates. Once downloaded they can be updated on a daily basis averaging about 580 kB. CRLite can be deployed by simply installing a plug-in on the browser. [11] However, when CRLite was proposed the bloom filter cascade was only 10 MB. But already a year after the proposal the filter used up 18.1 MB of space [10]. This is because between January 2017 and January 2020 the number of live certificates went up from 30 million to over 434 million. This is because of services like Let's Encrypt which enables automatic issuing for certificates, as well as efforts to normalize using only encrypted web traffic. [8]

6.2. Let's Revoke

Let's Revoke is an approach based on CRLite. But compared to CRLite, Let's Revoke needs 28% of network bandwidth. It utilizes a push-based model with a focus on minimizing network bandwidth consumption while maintaining a global revocation coverage. [8] To achieve this Let's Revoke invented a method using so-called Certificate Revocation Vectors (CRV), Revocation Numbers (RN), and Revocation IDs (RID). CRVs are dynamically-sized bit vectors. Each bit in the vector represents the revocation status of one specific certificate. The bits in the vector are mapped to their respective certificates with the RN. To limit the size of the vectors and ease the use, they are separated by date of expiration. RIDs are then used to identify which CRV a certificate belongs to. To deploy Let's Revoke, it is necessary to make adjustments on the client as well as on the CA side. Incremental deployment is also possible. [8]

6.3. CRT

A certificate revocation table (CRT) is a pull-based revocation approach. To make it work, a server maintaining a certificate working set is needed. The working set gets updated periodically by querying OCSP responders or CRL endpoints. The certificate information can be accessed by the clients by either downloading a file or an on-demand API. [10] CRT maintains a cache containing the revocation status of certificates with a high probability of usage. This is expected to make it more difficult for attackers to perform an attack and allows the client to hard-fail. CRTs can contain revoked and non-revoked certificates. When only used for revoked certificates it does not use a lot of bandwidth. The required bandwidth for a CRT is easily scalable, also over the next years, since it is only directly influenced by the number of certificates used by the client. Even during a mass revocation event, it would remain steady since the revocation status of every certificate is already in the working set of the CRT no matter if it is revoked or not. CRT allows being updated according to the needs of the consumer. It allows for the privacy of the client. A CRT would need to be deployed on the server by an administrator. CRT is still new and there are plans for improvements, which can be read about in [10].

6.4. OCSP Must-Staple

OCSP imposes a burden on the client and exposes the browsing behavior of the client to the CA responsible for issuing the certificate to the server. To combat these problems, OCSP Stapling got introduced. Using OCSP Stapling a server must periodically query an OCSP request to the CA and cache this signed response, proving the validity of the certificate. When a client then wants to establish a connection to the server, the server must send the signed response stapled with the certificate to the client during the TLS handshake. [3]
OCSP Stapling solves the initial problems of OCSP. However, the clients can still choose to continue connecting to the server if the OCSP response is not provided. Connecting without a signed response is called soft-failing. OCSP Must-Staple was invented to address this problem. To use OCSP Must-Staple certificates must include an OCSP Must-Staple extension. Available OCSP responders, certificates that support OCSP Must-Staple, and browsers that enforce the OCSP Must-Staple extension are required to implement OCSP Must-Staple. Thus, a change for every player involved in certificate revocation is necessary. However, in an experiment conducted 36.8% of OCSP Responders had at least one outage that lasted for hours, only 0.02% of certificates support OCSP Must-Staple and the only browser to implement OCSP Must-Staple so far is Firefox. [3] Additionally, an attacker could perform a DoS attack targeted at the OCSP responders, making the website inaccessible during this period to clients. Furthermore, OCSP Must-Staple had problems with CA inconsistencies and bugs in server implementation. [10]

6.5. Comparison

To evaluate and compare the different approaches the following section takes a look at them through six cate-

	CRLite	Let's Revoke	CRT	OCSP Must-Staple
Efficiency	18 MB & 580 kB/day	2 MB & 114 kB/day	6.71 MB & 205 kB/day	1.3 kB/TLS handshake
Privacy	yes	yes	yes	yes
Auditability	yes	yes	yes	yes
Timeliness	1-2 days	1-2 days	1-2 days	4 days
Deployability	High	Medium	Medium	High
Failure Model	Hard-Fail	Hard-Fail	Hard-Fail	Soft-Fail

TABLE 1: Comparison Of The Different Methods [8], [10]

gories of measurement [8], [10]:
Efficiency is defined by bandwidth consumption for the client.
Timeliness Measures in which intervals the methods get updated.
Failure Model Evaluates how the client behaves when unable to get the revocation status of a certificate.
Privacy Does the approach ensure the privacy of the client?
Deployability How high are the deployment requirements?
Auditability Is the client able to audit the result for the revocation check?
The comparison of the different methods is in table 1.

7. Evaluation

In table 1 you can see the values for the different methods. When looking at efficiency, the first value is the initial download and the second value is the daily update to maintain the set. OCSP Must-Staple is the only method not having an initial download, but it needs 1.3 kB for every TLS handshake performed. This would impose a bandwidth burden on the client. Let's Revoke seems to be the best when it comes to efficiency, but CRT also has the option of using only revoked certificates which would be considerably lower with 1.92 kB & 0.21 kB/day. None of the methods exposes the privacy of the client, and all of them are auditable. There is also no difference in the timeliness of the methods with OCSP Must-Staple as the exception, which gets updated only every four days. When it comes to deployability, both CRLite and OCSP Must-Staple demand more change and effort being integrated into the existing infrastructure. Let's Revoke, and CTR does not demand much change and can be deployed with only demanding change from two actors in certificate revocation less. All of the methods hard-fail. OCSP Must-Staple also should hard-fail in theory, but that did not hold true when it was tested in real life. Having the categories of measurement in mind one might consider that Let's Revoke is the best choice. This is because of the size of the downloadable files and the deployability. However, when it comes to scalability and the ability to deal with mass revocation CTR is the more sensible choice.

8. Conclusion

This paper introduced certificate revocation and an overview of today's status quo for revocation validation and potential methods to improve the said standard. First, it was assessed how well the three different actors involved handle certificate revocation. The conclusion was that there is no established standard and revoked certificates get completely ignored by mobile browsers and there is no certainty for desktop browsers. Mozilla Firefox and Google Chrome both abandoned CRL and started deploying their CRLs: OneCRL and CRLSet. However, these only cover a tiny fraction of the present revoked certificates. Server administrators must manually revoke certificates which is not a fail-proof system and opens up time-frames of vulnerabilities. In the last years, approaches and methods of handling this problem were proposed. We looked at a few of them and assessed which one is best for real-life deployment. There are three different types of approaches: Pull-based, push-based, and network-assisted. The methods introduced were: CRLite (push-based), CRT (pull-based), Let's Revoke (push-based) and OCSP Must-Staple (network-assisted). Comparing these approaches led us to the conclusion that Let's Revoke is the most suitable method for real-life deployment at the moment. However, CRT is still in active development and might be more suitable in the future

References

[1] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, G. Wilson, Christo Eason, B. Noble, and I. N. Sneddon, "An End-to-End Measurement of Certificate Revocation in the Web's PKI," *IMC '15: Proceedings of the 2015 Internet Measurement Conference*, pp. 183–196, 2015.

[2] K. Kiyawat, "Do Web Browsers Obey Best Practices When Validating Digital Certificates?" 2014.

[3] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson, "Is the web ready for ocsp must-staple?" *IMC '18: Proceedings of the Internet Measurement Conference 2018*, pp. 105–118, 2018.

[4] "What percentage of internet traffic is mobile?" https://www.oberlo.com/statistics/mobile-internet-traffic, accessed: 2021-03-22.

[5] "Usage share of operating systems," https://en.wikipedia.org/wiki/Usage_share_of_operating_systems, accessed: 2021-03-22.

[6] "Network security services," https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS, accessed: 2021-03-22.

[7] "Ca:revocationplan," https://wiki.allizom.org/CA:RevocationPlan, accessed: 2021-03-22.

[8] T. Smith, L. Dickinson, and K. Seamons, "Let's Revoke: Scalable Global Certificate Revocation," *Xu, Sadeghi (Hg.) 2020 – Proceedings 2020 Network and Distributed*, 2020.

[9] L. Zhang, D. Choffnes, D. Levin, T. Dumitras, A. Mislove, A. Schulman, and C. Wilson, "Analysis of SSL certificate reissues and revocations in the wake of heartbleed," *IMC '14: Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 489–502, 2014.

[10] L. Dickinson, T. Smith, and K. Seamons, "Leveraging locality of reference for certificate revocation," *ACSAC '19: Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 514–528, 2019.

[11] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers," *2017 IEEE Symposium on Security 52017*, pp. 539–5556, 2017.

Optimizations for Secure Multiparty Computation Protocols

Leilani Tam von Burg, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: leilani.tam-von-burg@tum.de, christopher.harth-kitzerow@outlook.de

***Abstract*—The BGW protocol is a protocol for secure multiparty computation based on Shamir's secret sharing scheme. It allows the computation of functions by representing them as arithmetic circuits composed of addition and multiplication gates. Many steps of the protocol are quite efficient as they do not require encryption or communication. However, multiplication gates require communication and impact the efficiency of the protocol negatively. Therefore, optimization techniques improving the multiplication operation have been developed. In this paper, we focus on the optimization technique of Beaver triples.**

***Index Terms*—BGW Protocol, Shamir's secret sharing, Beaver Triples**

1. Introduction

There exist many applications where there is a need for computations that keep the inputs secret. Examples of this are secure auctions, voting, secure machine learning or computations on databases that hold private information. Secure Multiparty Computation Protocols do exactly this. They enable a joint computation on a group of parties without disclosing the private inputs of the participants [1].

The idea of Secure Multiparty Computation Protocols was first introduced by Yao in the 1980s [1], [2]. He illustrated the necessity for this type of computation with the two millionaires problem. Here, two millionaires want to determine who is richer without disclosing their individual wealth. Yao's protocol is based mainly on garbled circuits [2].

Later, different protocols were developed to expand from two party to multiparty computation and improve on efficiency. The implementation of Fairplay in 2004 is considered the first proper implementation of such a protocol [3]. Admittedly, its scalability and performance was very limited. Today, more efficient implementations exist and practical uses are becoming more and more common. This paper introduces the BGW protocol and one possible optimization technique often implemented in combination.

2. BGW Protocol for Secure Multiparty Computation

The BGW protocol was introduced by Ben-Or, Goldwasser and Wigderson [4]. It differs from other well known protocols in that it is not based on garbled circuits. Instead, it is based on Shamir secret sharing [5]. The protocol can compute any function f over a field F by representing the function as an arithmetic circuit composed of addition, multiplication and multiplication-by-constant gates. Multiplication by a constant can be represented by an addition and will therefore be omitted in the further discussion. The evaluation of the circuit is done gate-by-gate.

Since we require the computation to be secure, each input wire is only known by one party that desires to keep it private. Additionally, no intermediate values should be revealed during the evaluation of the circuit. In order to achieve the desired privacy during the evaluation of the function, the value of each wire is kept secret by hiding it in a polynomial of degree t which is shared between the parties.

2.1. Shamir's Secret Sharing

Since secret sharing is a fundamental part of the BGW protocol, it will be shortly introduced. The idea behind secret sharing can be illustrated by the following example. Imagine a treasure chest which requires multiple keys to be opened. These keys are held by different parties. Therefore, the treasure can only be accessed when the parties come together to unlock the chest.

The protocol can be split into two phases: a sharing phase and a reconstruction phase. During the sharing phase, the secret s is split into shares held by the different parties. Firstly, the secret s must be "locked in the chest". This is done by encrypting the secret in a polynomial of the following form.

$$f(x) = a_t x^t + ... + a_1 x + a_0 \tag{1}$$

where $a_0 = s$ and $a_t, ..., a_1$ are random coefficients, such that $f(0) = s$. Each party P_i then receives one point $(\alpha_i, [f(\alpha_i)])$ on the polynomial, which we refer to as its share. This value can be interpreted as the "key" held by that specific party. For clarity, we will refer to shares by using square brackets throughout the paper.

The threshold t is chosen such that it corresponds to the assumed maximum number of faulty parties. Therefore, $t + 1$ points are required to reconstruct the secret by interpolating the polynomial. Less points will not reveal the secret. This corresponds to the reconstruction phase.

2.2. Security scenarios for the BGW protocol

When choosing a threshold value t for the BGW protocol, we differentiate between two different types of faulty parties.

Semi-honest security A semi-honest adversary is considered honest-but-curious. This means it follows the protocol honestly but it may try to learn as much information as possible during the execution. Therefore, we consider it a passive adversary. This includes parties colluding and pooling their information together in order to learn as much as possible [1].

For every n-ary function $f(x_1, ..., x_n)$, there exists a protocol for computing f with perfect security in the presence of a semi-honest adversary controlling $t < n/2$ parties. This means we require an honest majority in this case [4]. Evidently, we cannot allow any adversaries in a two party computation. In this situation, the secret inputs are encrypted with linear functions. Therefore, knowing the gradient allows direct reconstruction of the secret.

Malicious security A malicious adversary is active, which means it can take any action it desires and deviate from the protocol. Therefore, it can provide any input it wants as well, which can affect the honest parties inputs. A malicious adversary can control up to $t < n/3$ parties while ensuring perfect security [1].

It should be noted here, that the BGW protocol is secure from an information-theoretic standpoint when adhering to the above choices for the threshold t [4]. It does not rely on cryptographic assumptions. This means that the protocol is secure for adversaries with unlimited computing power. For example, quantum computers do not pose a threat to the security.

2.3. BGW Protocol

The BGW protocol can be subdivided into three main phases:

1) Input sharing phase
2) Computation of the circuit gate-by-gate (additions, multiplications)
3) Output Reconstruction phase

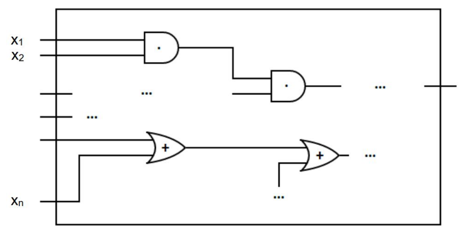


Figure 1: Example of a function represented by an arithmetic circuit with inputs $x_1, ..., x_n$. The input values as well as the intermediate values are secret shared to ensure their privacy over the entirety of the circuit.

2.3.1. Input sharing phase. The input sharing phase of the BGW protocol follows the Shamir secret sharing scheme. Each party P_i encrypts its input x_i in a random polynomial $f_i(x)$ of degree t, where $f_i(0) = x_i$ and then sends each party P_j a share $[f_i(\alpha_j)]$. This way, all parties obtain shares of the other parties inputs.

2.3.2. Computation of the circuit. At each gate, the parties compute the shares of the output wire using the shares of the input wires. The intermediate values stay hidden throughout the circuit.

Addition Gates The computation of addition gates is inexpensive since it does not require communication. Let a and b be the input values and $g_a(x)$ and $g_b(x)$ be the polynomials hiding the input values according to the secret sharing scheme. $g_a(x) + g_b(x) = h_{a+b}(x)$ is the operation at the gate and $\alpha_1, ..., \alpha_n$ are the interpolation points of the individual parties shares. Initially, each party P_i holds the shares $[g_a(\alpha_i)]$ and $[g_b(\alpha_i)]$ of the input which it can add locally. Then, each party holds a share $[h_{a+b}(\alpha_i)]$ of the output $h_{a+b}(x)$. This share of the output can be used directly as the input at the next gate, since there is no need for communication during addition operations. Computations can be done locally on the shares throughout the circuit (as long as they are all additions) until the final output, where the shares are combined to recover the sum. To do so, the constant term $h_{a+b}(0) = a + b$ is evaluated. The output polynomial is still of degree t.

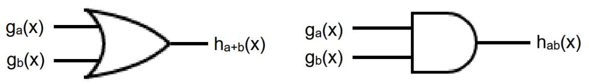


Figure 2: Illustration of an addition gate on the left and a multiplication gate on the right with the secret shared wire inputs $g_a(x)$ and $g_b(x)$ of the input values a and b.

Multiplication Gates The computation of multiplication gates is not as straight forward. Simply multiplying the shares locally as it is done for addition causes two problems. Firstly, the degree becomes $2t$ after a single computation. Multiple multiplication gates cause the degree of the polynomial to become too large. There are not enough interpolation points to recover the result anymore. Additionally, the product of two random polynomials is not fully random anymore. We need a way to compute the multiplication while keeping the polynomial at degree t and ensuring it stays random. This is referred to as degree reduction and randomization [4].

Degree reduction and randomization Assume input values a and b. Degree reduction relies on following property of polynomials:

For any polynomial $h(x)$ with degree $t < n$, there exist constants $\lambda_1, ..., \lambda_n$ and interpolation points $\alpha_1, \alpha_2, ..., \alpha_n$ such that:

$$h(x) = \lambda_1[h(\alpha_1)] + ... + \lambda_n[h(\alpha_n)]$$
$$h(0) = ab$$

That is, we can represent the result of the multiplication by a linear combination $h(x) = \sum_{i=1}^{2t} \lambda_i [h(\alpha_i)]$ of the parties shares.

This can be illustrated more thoroughly by observing the following equations. Let $h(x) = h_{2t}x^{2t} + ... + h_1 x + ab$ be a polynomial of degree $2t$ hiding the secret $h(0) = ab$. In equation (2), we multiply an invertible Vandermonde matrix with the coefficients of the polynomial. This operation corresponds to the evaluations of the polynomial $h(x)$ on the interpolation points $\alpha_1, \alpha_2, ..., \alpha_n$.

$$\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & ... & \alpha_1^{2t} \\ 1 & \alpha_2 & \alpha_2^2 & ... & \alpha_2^{2t} \\ \vdots & & & & \\ 1 & \alpha_n & \alpha_n^2 & ... & \alpha_n^{2t} \end{bmatrix} \begin{bmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \end{bmatrix} = \begin{bmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{bmatrix} \tag{2}$$

However, since our goal is to compute our secret ab with each parties shares $[h(\alpha_i)]$, we must invert the matrix. A simple way to invert the Vandermonde matrix is explained in [6]. For example, a Vandermonde matrix

$$V = \begin{bmatrix} 1 & c_1 & c_1^2 & c_1^3 \\ 1 & c_2 & c_2^2 & c_2^3 \\ 1 & c_3 & c_3^2 & c_3^3 \\ 1 & c_4 & c_4^2 & c_4^3 \end{bmatrix} \tag{3}$$

has the inverse V^{-1}=

$$\begin{pmatrix} \frac{c_2c_3c_4}{(c_4-c_1)(c_3-c_1)(c_2-c_1)} & \frac{-c_1c_3c_4}{(c_4-c_2)(c_3-c_2)(c_2-c_1)} & \frac{c_1c_2c_4}{(c_4-c_3)(c_3-c_2)(c_3-c_1)} & \frac{-c_1c_2c_3}{(c_4-c_3)(c_4-c_2)(c_4-c_1)} \\ \frac{-c_2c_3+c_2c_4+c_3c_4}{(c_4-c_1)(c_3-c_1)(c_2-c_1)} & \frac{c_1c_3+c_1c_4+c_3c_4}{(c_4-c_2)(c_3-c_2)(c_2-c_1)} & \frac{-c_1c_2+c_1c_4+c_2c_4}{(c_4-c_3)(c_3-c_2)(c_3-c_1)} & \frac{c_1c_2+c_1c_3+c_2c_3}{(c_4-c_3)(c_4-c_2)(c_4-c_1)} \\ \frac{c_2+c_3+c_4}{(c_4-c_1)(c_3-c_1)(c_2-c_1)} & \frac{-c_1-c_3-c_4}{(c_4-c_2)(c_3-c_2)(c_2-c_1)} & \frac{c_1+c_2+c_4}{(c_4-c_3)(c_3-c_2)(c_3-c_1)} & \frac{-c_1-c_2-c_3}{(c_4-c_3)(c_4-c_2)(c_4-c_1)} \\ \frac{-1}{(c_4-c_1)(c_3-c_1)(c_2-c_1)} & \frac{1}{(c_4-c_2)(c_3-c_2)(c_2-c_1)} & \frac{-1}{(c_4-c_3)(c_3-c_2)(c_3-c_1)} & \frac{1}{(c_4-c_3)(c_4-c_2)(c_4-c_1)} \end{pmatrix}$$

This shows that in our case, the entries of the inverted matrix depend only on α_i, the points where the function is evaluated and which are public. We introduce the values λ_i corresponding to each of the individual entries of the first line of the inverted matrix.

$$\begin{bmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \end{bmatrix} = \begin{bmatrix} \lambda_1 & \dots & \lambda_n \\ \vdots & & \\ \dots & & \end{bmatrix} \begin{bmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{bmatrix} \tag{4}$$

We are only interested in the first line of (3) since the only coefficient of interest is the secret ab. This corresponds exactly to the linear combination introduced earlier.

The protocol followed to compute the multiplication is the following.

- Each party P_i computes its share
$$[h(\alpha_i)] := [g_a(\alpha_i)][g_b(\alpha_i)] \tag{5}$$
locally.
- $[h(\alpha_i)]$ is then secret shared with the other parties using a degree t polynomial $H_i(x)$
- Now, each party holds a share $[H_1(\alpha_i)], ..., [H_n(\alpha_i)]$ of each other parties polynomial $H_i(x)$. Each party can compute the output
$$[H(\alpha_i)] = \lambda_1[H_1(\alpha_i)] + ... + \lambda_n[H_n(\alpha_i)] \tag{6}$$
locally.
- Each party P_i now holds a share $[H(\alpha_i)]$ of $H(x) := \lambda_1 H_1(x) + ... + \lambda_n H_n(x)$.

It is important to note that, because each $H_1(x), ..., H_n(x)$ is of degree t, $H(x)$ is again of degree t. Additionally, each $H_1(x), ..., H_n(x)$ is random since they are created by following the secret sharing scheme where all coefficients are random. Therefore, $H(x)$ is also random, since the addition of random functions is still random.

We have achieved a dimensionality reduction and ensured the randomness of the polynomial as desired. Unfortunately, the protocol for the computation of multiplications requires communication. This is inefficient. Therefore, optimization techniques such as Beaver triples have been developed to alleviate the communication costs. This will be further explained in the next section.

Output Reconstruction phase The output of a multiplication can be computed by evaluating

$$\begin{aligned} H(0) &= \lambda_1 H_1(0) + ... + \lambda_n H_n(0) \\ &= \lambda_1[h(\alpha_1)] + ... + \lambda_n[h(\alpha_n)] \\ &= ab \end{aligned}$$

3. Beaver Triples as Optimization Technique

The majority of the cost of the BGW protocol is caused by the communication required for multiplication operations. Ideally, a portion of the cost would be moved to the pre-processing phase. Unfortunately, the operations are dependent on the circuit inputs, which only become available during the online phase. Beaver triples allow for a way to move the majority of the communication to the pre-processing phase given even these circumstances. The basic idea is that the parties produce shared data during the offline phase that does not require information on the inputs [1].

A Beaver triple, also known as multiplication triple is a triple of three secret shared values $[a], [b], [c]$ where a, b are uniform random values unknown to all parties and $c = ab$.

3.1. Generation of the triples

There are different ways of generating the multiplication triples. A first option is trusted party generation. This protocol requires an honest third party that samples the triple (a, b, c) and distributes shares to the parties participating in the computation. The third party does not participate in the actual computation at all and does not have to be trusted with the actual inputs. However, it must compute the triples correctly and refrain from sharing their values. A second option is based on oblivious transfer which shall be shortly introduced.

Oblivious Transfer A sender S holds two secrets x_0, x_1 and a receiver R holds a choice bit $b \in \{0,1\}$. The receiver learns x_b while staying oblivious to the other secret x_{1-b} and the sender does not learn anything about if or what information was transferred.

In the following, an example of generating triples based on oblivious transfer is illustrated. Say Alice and Bob want to generate a beaver triple. Firstly, Alice randomly samples values (x_A, y_A) and r_A and Bob randomly samples values (x_B, y_B) and r_B.

Next, Alice acts as the sender in an oblivious transfer with the input pair $(r_A, x_A \oplus r_A)$. Bob acts as the receiver using y_B as the selection bit. If $y_B = 0$, he learns r_A, else, he learns $x_A \oplus r_A$. In total, he learns $x_A y_B \oplus r_A$. The same thing is done in the other direction, so Alice learns $x_B y_A \oplus r_B$.

Now, Alice computes

$$z_A \leftarrow r_A \oplus x_A y_A \oplus x_B y_A \oplus r_B \tag{7}$$

and Bob computes

$$z_B \leftarrow r_B \oplus x_B y_B \oplus x_A y_B \oplus r_A \tag{8}$$

This results in

$$z_A \oplus z_B = (x_A \oplus x_B)(y_A \oplus y_B) \tag{9}$$

. Therefore, $(x_A, x_B), (y_A, y_B), (z_A, z_B)$ correspond to a Beaver triple.

Another option for the generation of the triples is based on homomorphic encryption [7], a way of computing on encrypted data without having to decrypt it.

3.2. Computing multiplications with Beaver triples

During the online step, the beaver triples are used during computation to diminish the necessary communication. Assume we generated a triple (a, b, c) using one of the generation methods. Additionally, we have the two input values α, β that we would like to securely multiply with each other. The parties hold secret shares of the input values. We define these as $[v_\alpha]$ and $[v_\beta]$. The computation with the beaver triples follows the following protocol [1].

1) Each party computes $[v_\alpha - a]$ and $[v_\beta - a]$ locally. Then, all parties publicly announce their shares in the form $d = v_\alpha - a$ and $e = v_\beta - b$ (the secret values v_α and v_β are hidden by a and b).

2) Following equality holds:

$$\begin{aligned} v_\alpha v_\beta =& (v_\alpha - a + a)(v_\beta - b + b) \\ =& (d + a)(e + b) \\ =& de + db + ae + ab \\ =& de + db + ae + c \end{aligned}$$

A share of $[v_\alpha v_\beta] = de + d[b] + e[a] + [c]$ is computed locally by each party.

Therefore, each party must only broadcast two values d and e per multiplication. This is much more cost effective than the communication required in plain BGW to compute multiplications.

4. Efficiency of BGW Protocol

Since, the BGW protocol uses secret sharing to hide its inputs, it does not rely on encryption. Generally, this is considered more efficient than a cryptographic approach. But as we have seen, certain operations involve complications and communication that strongly impact the efficiency.

The BGW protocol is very efficient for arithmetic circuits containing mostly additions [8]. Unfortunately, it is not ideal for functions requiring many multiplications. As a rule of thumb, we assume communication is much more expensive than computation and decryption. This means that tasks like matrix multiplications are difficult to solve with the BGW protocol. This would require many multiplication gates and a very large cost related to communication. For example, neural networks require extensive matrix multiplications and are not ideal for the BGW protocol.

Additionally, certain operations are expensive to represent as arithmetic circuits. Arithmetic circuits operate over a finite field F that must be set in advance and be large enough to prevent overflow [9]. In order to compute operations such as comparisons, bit-shifts and equality tests, a bit-decomposition is required. This conversion is expensive. Therefore, these are also operations that should be avoided with the BGW protocol.

5. Conclusion

This paper provided insight into the functionality of the BGW protocol for secure multiparty computation. Perhaps the most interesting component of this protocol is the degree reduction step, necessary to allow the computation of multiplication gates in a secure way. Unfortunately, this step also has significant negative impact on the efficiency of the protocol. This is why optimization techniques have been implemented to alleviate this impact, such as the Beaver triples introduced in this paper. All in all, the BGW protocol is often more efficient than protocols relying on a cryptographic approach. This depends on the type of function being evaluated. In general, operations based on many multiplications might be more efficiently computed with a different protocol.

References

[1] D. Evans, V. Kolesnikov, and M. Rosulek, *A Pragmatic Introduction to Secure Multi-Party Computation*, 2018, vol. 2, no. 2-3. [Online]. Available: http://dx.doi.org/10.1561/3300000019

[2] A. C. Yao, “Protocols for secure computations,” pp. 160–164, 1982.

[3] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, “Fairplay — a secure two-party computation system,” 06 2004.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC ’88. New York, NY, USA: Association for Computing Machinery, 1988, p. 1–10.

[5] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979.

[6] E. Rawashdeh, “A simple method for finding the inverse matrix of vandermonde matrix,” 01 2020.

[7] S. University, “Cs 355: Topics in cryptography.” [Online]. Available: https://crypto.stanford.edu/cs355/18sp/lec7.pdf

[8] T. Rabin, “Secure multiparty computation,” 2014. [Online]. Available: https://www.youtube.com/watch?v=NOtsxHoIcWQ&t=618s&ab_channel=Technion

[9] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, “Sok: General purpose compilers for secure multi-party computation,” in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1220–1237.

Taxonomy of the Performance of P4 Targets

Irina Tsareva, Dominik Scholz*, Sebastian Gallenmüller*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: irina.tsareva@tum.de, {scholz,gallenmu}@net.in.tum.de

***Abstract*—The P4 language enables abstraction and flexible programming of the data plane for various hardware- or software-based targets, like field programmable gate arrays (FPGAs) or software switches for general-purpose CPUs. Each of these targets has its own limitations in available parallelization (e.g., multi-core), memory resources (e.g., RAM), or number of high bandwidth ports. Thus, one P4 program may have different latency and throughput characteristics, depending on the target machine. Analyzing the performance differences for P4 targets is crucial to identify bottlenecks and predict the performance of any P4 program. In this paper, we provide a comparison of performance measurements of P4 targets and show the impact of different P4 constructs. Hardware-based solutions have the lowest latency and highest throughput. Furthermore, we describe two model approaches to predict the performance of P4 targets: models based on benchmarking and on stochastics. While benchmarking-based models allow a straight performance comparison, they are highly dependent on target-specific information, and thus, may not be applicable to every target. Whereas the probabilistic model is implemented to be more general and can be further refined with information about the target.**

***Index Terms*—programmable data plane, P4, performance, performance model**

1. Introduction

Software-defined networking (SDN) [1,2] enables faster deployment, centralized management, and scalability through network control. This architecture decouples the control and data plane physically, such that the software-based control plane controls the data plane (e.g., switches, routers) over an open protocol like OpenFlow. Consequently, to support new header fields the OpenFlow API specification has to be extended and hardware switches redesigned or reprogrammed (e.g., application-specific integrated circuits (ASICs)), leading to an increased complexity of the API, additional deployment cycles, and a need for domain experts. The programming protocol-independent packet processors (P4) language aims to solve these issues by providing an abstraction layer between an API such as OpenFlow and the data plane.

P4 [2] is a protocol-independent, target-independent, and field reconfigurable language that can express how the data plane has to process packets. Protocol-independent means that there is no set of supported protocols (e.g., IPv4, Ethernet), and thus, this set has to be defined as desired. With P4, different supported platforms (*P4 targets*) can be programmed, like field programmable gate arrays (FPGAs), ASICs, or network processing units (NPUs). Field reconfigurable describes the property that P4 targets can be reconfigured after they were shipped.

Various implementations of P4 targets exist; either software- or hardware-based. Although the hardware-based solutions provide high bandwidth ports and specialized many-core architectures, they differ highly in their price, e.g., about 7,000$ for a NetFPGA SUME board [3], and maintenance costs. Moreover, each implementation uses different libraries, compilers, and optimization techniques to enable different features (e.g, stateful operations), and thus, are not suitable for every network setup. Yet, the implementation also influences the latency or throughput of P4 programs. It is crucial to understand and estimate the performance of P4 programs on a P4 platform correctly to have predictable execution and to identify optimization possibilities of a given setup.

At present, two approaches for performance estimation of arbitrary P4 programs exist: models based on benchmarking and on stochastics. The goal of this paper is to provide an overview of both approaches and their implementations. To this end, we focus on a selection of P4 targets (Section 2) and describe necessary performance measurements for the benchmarking-based model (Section 3). We present implementations of the benchmarking-based approach and one probabilistic model and discuss their advantages and disadvantages (Section 4).

2. Background

A device must explicitly support P4 programmability, otherwise it is not possible to program the device via P4. To understand the architecture of a *P4 target*, knowledge about the P4 language is needed.

2.1. P4 Programming Language

A P4 program consists of three stages: the parser, pipeline, and deparser stage. The parser is a finite state machine that takes the arriving packets (*ingress queue*) and extracts its headers into a stack. The pipeline includes multiple match-action units that process the parsed headers by performing user-defined actions such as modifications upon them. The match-type can be either exact, ternary, or lpm. Finally, the deparser reconstructs the packets with the modified, added, or removed headers and sends them to the outgoing packet buffer (*egress* *queue*) or drops them. In this context, we call the parsers, actions, and tables *P4 constructs*. They are the basic blocks out of which arbitrary P4 programs can be build. A P4 programs defines network functions (NFs), like NATs or firewalls. [4]

The *P4 compiler* consists of two parts: A generic open-source front-end and a target-specific back-end compiler (provided by the vendor). The front-end compiler transforms the P4 program into an intermediate representation (IR), which is then mapped into target-specific code (e.g., in C or Verilog) by the back-end compiler. [4]

P4 targets implement a target-specific *P4 architecture model* which describes the interface as well as fixed-function and programmable blocks. This interface enables the P4 constructs to be mapped onto the target. [5]

2.2. P4 Targets

We present a selection of P4 targets ordered by their degree of flexibility (descending), specialization (ascending), and price (ascending). Although GPU-based implementations exist [6], we do not discuss them here, since not enough performance measurement results exist.

2.2.1. Software-based. Software-based implementations can be run on a general-purpose CPU. They can make use of, e.g., the heap memory and available cores. They are more flexible to implement, since there are hardly any hardware constraints, but their behavior might be non-deterministic due to scheduling or interrupts during runtime [7].

Behavioral Model version 2 (bmv2) – bmv2 [8] is the reference software switch implementation of `p4.org`. This switch has only developing, testing, and debugging purposes due to its high latency and low throughput (up to 1 Gbit/s).

T4P4S DPDK-based – The data plane development kit (DPDK) is a collection of user-space libraries to accelerate packet processing. It includes multiple features to accelerate software-switches. The P4 program is compiled through T4P4S into C code that can be run on top of DPDK. [4]

PISCES – PISCES [9] is a software switch that is based on the virtual Open vSwitch (OVS) and uses the DPDK fast path instead of the less performant kernel modules. Additionally, the authors added new primitives to support encapsulation.

BPF-based – P4rt-OVS [10] extends the OVS-DPDK by the user-space berkeley packet filter (uBPF). These libraries add support for runtime extensibility and stateful operations. This implementation introduces a new front-end P4-to-uBPF compiler.

2.2.2. NPU-based. NPUs consist of "tens of multi-threaded purpose-built" cores that are optimized for network data packet processing, and thus, they are more specialized than CPUs. One example is the **Netronome SmartNIC**. To program this NPU, the IR is compiled into C code, and then, into firmware which is loaded onto the P4 target. [4]

2.2.3. FPGA-based. Hardware design can be described using a hardware descriptive language (HDL). This design is then synthesized, placed, and routed onto a specific FPGA. While developing the design the developer already has to know the target FPGA so she can meet platform-specific constraints, such as timing or available resources. If the constraints are not met, placing and routing will fail. In case of success, a bitstream can be generated and flashed onto the FPGA. The advantage is that the hardware can be optimized for an application.

P4→NetFPGA – P4→NetFPGA [11] is a workflow on top of the Xilinx P4-SDNet compiler and NetFPGA SUME to compile P4 code into Verilog code. The target FPGA is the NetFPGA SUME board.

P4-to-VHDL – P4-to-VHDL [12] compiles a P4 program into VHDL code without having an IR.

P4FPGA – P4FPGA [13] extends the `p4.org` front-end compiler by a back-end that first generates Bluespec System Verilog code. Then, this code is converted into Verilog code that can be synthesized to either Xilinx or Altera FPGAs. The generated code contains a P4 programmable packet-processing pipeline and a fixed-function pipeline.

2.2.4. ASIC-based. Hardware switches that are hard-wired for a specific application are ASIC-based. Their function cannot be reprogrammed making them less flexible, but more specialized. **Intel Barefoot Tofino 1** [14] is such an Ethernet switch. It uses static RAM (SRAM) and ternary content-addressable memory (TCAM) for P4 tables, depending on the match types [15].

3. Performance of P4 Targets

In this section we compare the performance of a selection of P4 targets on basic P4 constructs. Performance metrices are latency and throughput. Typically, the latency and throughput are best for an FPGA and ASIC, since they have application-specific, optimized hardware. However, hardware constraints limit their expressibility.

3.1. Latency

The overall latency depends on the amount of occurrences of basic P4 constructs in a P4 program. Table 1 depicts the impact of eight P4 constructs on the bmv2, T4P4S switch, PISCES, Netronome SmartNIC, and NetFPGA SUME (compiled via P4→NetFPGA).

For instance, modifying header fields has a negligible impact on the T4P4S switch, Netronome SmartNIC, and NetFPGA SUME board since they write the complete header, even if a single field is changed [4]. Since whole headers are emitted in the P4 deparser syntax [4] we expect to see a similar behavior for bmv2 and PISCES.

Comparing the targets listed in Table 1, the NetFPGA SUME board has the lowest overall latency for each P4 construct, while bmv2 has the highest. The T4P4S switch has comparable latency to the Netronome SmartNIC with packet sizes of 256 Bytes (B); for packets larger than 1000B or 1500B the Netronome SmartNIC has a better latency by 3ns-7ns (depending on the amount of the occurrence of P4 constructs). However, the Netronome SmartNIC scales the worst with increasing pipeline and action complexity. [4]

P4 constructs	**Impact on Targets**								
	bmv2 [7]		T4P4S Switch [4]	PISCES [7]		Netronome SmartNIC [4]		NetFPGA SUME (P4→NetFPGA) [4]	
Parsing Headers	- -	$O(n^2)$	+	-	$O(n)$	-		- -	$O(n)$
Modifying Header Fields	n.a.		+	n.a.		+		+	
Operation Executions	- -	$O(n)$	n.a.	- -	$O(n)$	+		n.a.	
Modifying Headers	n.a.		+	n.a.		- -	$O(n)$	+	
Copying Headers	n.a.		+	n.a.		-		+	
Removing Headers	n.a.		+ +	n.a.		- -	$O(n)$	+ +	
Adding Headers	n.a.		-	n.a.		- -	$O(n^2)$	-	
Adding Tables	- -	$O(n^2)$	+	+		- -	$O(n)$	-	$O(n)$

TABLE 1: Latency impact of P4 constructs on P4 targets (median values). *(- - significant degradation (2.5μs – >30.3μs), - degradation (1.5μs–2.5μs), + no or a slight impact (0μs–1.5μs), ++ improvement (0.1μs–2μs), n.a. no value available)*

Osiński *et al.* [10] show that the latency of P4rt-OVS increases lineary for increasing number of match-action tables and is constant for varying number of table entries.

Vörös *et al.* [16] demonstrate that their implementation of T4P4S has comparable latency as PISCES. However, they do not compare it to isolated P4 constructs.

On the other hand, compiler and hardware design optimizations further decrease the overall latency. For example, P4rt-OVS [10], PISCES [9] and P4FPGA [13] implement post-pipeline editing, which postpones the modification of packets to the deparser. This reduces the performance at the deparser stage. PISCES merges multiple match-action pipelines into one, which leads to non-changing latency for increasing number of tables. P4-to-VHDL [12] introduces offset width and multiplexer optimizations. However, the authors do not provide performance comparisons with other P4 targets which is why we cannot classify its performance relative to the listed P4 targets. Zhou *et al.* [17] reduce the latency of Netronome SmartNIC and bmv2, by chaining NFs, reducing redundant functions, and bypassing undesired functions.

3.2. Throughput

Parallelism, like introduced by FPGAs, may significantly increase the throughput. P4FPGA outperforms PISCES also in terms of throughput. Moreover, throughput may depend on the complexity of the P4 program: If the number of headers increases, the throughput of P4FPGA (on NetFPGA SUME) and PISCES (on a CPU) decreases. [13]

PISCES [9] (optimized) has a smaller throughput by 2% compared to OVS. If new protocols are added, the throughput decreases, e.g., about 35% from 51.1Gbps to 33.2 Gbps if post-pipeline editing is activated.

Osiński *et al.* [10] show for three network functions, that the throughput of P4rt-OVS is comparable to PISCES and OVS for packet sizes of 128B and 256B; For 512B the throughput is larger than for PISCES.

P4-to-VHDL [12] can parse traffic with a complex protocol structure with 100 Gbps. The optimized Intel Barefoot Tofino 1 [14] can achieve a bandwidth of 100 Gigabit Ethernet per port.

4. Performance Models

Evaluating each possible P4 application and combination of P4 constructs for each P4 target is neither feasible nor desirable. A performance model should describe the estimated performance of an arbitrary complex P4 program for every or a specific P4 target. Next, we describe two approaches to model estimation and discuss their advantages and disadvantages.

4.1. Models Based on Benchmarking

This approach combines performance measurements of isolated P4 constructs obtained through benchmarking with the occurrences of P4 constructs obtained through P4 program analysis. Since the benchmarks do not test for real-world workloads but for core features of P4, they are called *synthetic benchmarks*. Each feature is tested in isolation with varying parameters, e.g., incrementally increasing number of packet headers and fields in the parser.

WhipperSnapper [7] was the first benchmarking suite for the P4 language. It contains latency, throughput, and memory usage measurements against five features: Parsing, processing, state accesses, packet modification, and action complexity. Additionally, WhipperSnapper includes target-dependent benchmarks, that test specifically for the hardware support and usage of ASICs and FPGAs as well as the latency and throughput of CPUs and NPUs with reduced scheduling and locking impact.

Harkous *et al.* [4,5] continue this idea and focus on eight features related to parsing, processing, packet modification, and action complexity (see **P4 constructs** in Table 1). Yet, their benchmarks focus only on latency. They explain and validate their model examplary for the Netronome SmartNIC, NetFPGA SUME board (compiled via P4→NetFPGA), and T4P4S switch. Each of these has target-specific latency measurement results. The slopes of the measurement results for each feature can be (piecewise) interpolated and stored in a target-profile-vector. This vector is calculated only once for each target since it does not change. Next, the P4 program has to be analyzed and the occurence of P4 constructs determined. The estimated average latency of a network function is then the sum of the prior measured latencies of the (isolated) P4 constructs with respect to their amount of occurrences.

Scholz *et al.* [15] suggest to derive models based on the weaknesses of a target. In the case of software switches, the weakness is their significantly varying latency due to different implementations of P4 elements (e.g., the match types) and the underlying memory access pattern. Whereas the resource utilization is a weakness of

ASICs since their available memory limit the complexity of P4 programs. The authors derive cost functions for varying properties of the match-action pipeline (e.g., table entry size and varying table key width lengths) for the T4P4S switch and Intel Barefoot Tofino 1.

All described models require target specific information (e.g., memory resources, software implementation) to accurately estimate the performance. For instance, the model of Harkous *et al.* [4] has an accuracy of more than 94% and is especially accurate for the NetFPGA SUME. It is less accurate for the Netronome SmartNIC due to its high dependency on the P4 pipeline. Moreover, latency measurements of isolated P4 constructs are not always additive; the measured latency of combined P4 constructs may be smaller due to e.g., reduced memory access [5]. Therefore, summing the latency of isolated P4 constructs may reduce the accuracy, too.

Due to the highly target-dependent nature of these models, the performance measurements for isolated P4 constructs could be made publicly available by researches, and hence, facilitate and accelerate performance prediction. The P4 program analysis can be done by the compiler. [4]

4.2. Models Based on Stochastics

While the previously described approach requires benchmarking to derive target-specific parameter values for cost functions, this approach is based on a more generic probabilistic model (cf. Bayesian network). The match-action tables of a P4 program are converted into a control flow graph (CFG). The CFG depicts all possible execution paths of the program: a node depicts the line number the program counter points to, while an edge is a possible transition to the next event; each node is associated with an execution cost. Thus, the expected execution cost of a P4 program is the sum of the conditional expected costs of an execution path. For instance, if a path is not executed, it has cost of 0. [18]

The advantage of this framework is that it is generic enough to be used for every target. To get more accurate results, additional information about the target (e.g., hardware configuration, runtime environment) can be added in a modular way. [18]

5. Conclusion and Future Work

The P4 language allows abstraction and programmability of the data plane. Due to its target-independence, multiple targets can be programmed using the same set of P4 constructs. However, the performance differ significantly for each target due to hardware constraints and software implementations.

In this paper we summarized studies comparing and analyzing the impact of different P4 constructs on the latency, throughput, and memory usage. These benchmarks in addition with information about the target implementation and the control flow can be used to derive performance models. Another approach is to use a probabilistic model based on expected execution costs. These performance models can estimate the performance for arbitrary complex P4 programs, and thus, help create predictable networks.

As future work, it would be of interest to analyze other P4 targets (e.g., P4rt-OVS) with respect to P4 constructs and compare these results with the existing ones. Furthermore, more exhaustive measurements could be done to also investigate the impact of not yet included performance aspects, such as jitter or power draw.

References

[1] IBM, “What is Software-Defined Networking (SDN)?” https://www.ibm.com/services/network/sdn-versus-traditional-networking, [Online; accessed 22-March-2021].

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-Independent Packet Processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[3] Digilent, “NetFPGA-SUME Virtex-7 FPGA Development Board,” https://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/, [Online; accessed 08-May-2021].

[4] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, “P8: P4 with Predictable Packet Processing Performance,” *IEEE Transactions on Network and Service Management*, pp. 1–14, 2020.

[5] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, “Towards Understanding the Performance of P4 Programmable Hardware,” in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–6.

[6] P. Li and Y. Luo, “P4GPU: Accelerate packet processing of a P4 program with a CPU-GPU heterogeneous architecture,” in *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2016, pp. 125–126.

[7] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, “Whippersnapper: A P4 Language Benchmark Suite,” in *Proceedings of the Symposium on SDN Research*, ser. SOSR ’17. Association for Computing Machinery, 2017, pp. 95–101. [Online]. Available: https://doi.org/10.1145/3050220.3050231

[8] p4.org, “BEHAVIORAL MODEL (bmv2),” https://github.com/p4lang/behavioral-model, [Online; accessed 22-March-2021].

[9] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, “PISCES: A Programmable, Protocol-Independent Software Switch,” in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM ’16. Association for Computing Machinery, 2016, pp. 525–538. [Online]. Available: https://doi.org/10.1145/2934872.2934886

[10] T. Osiński, H. Tarasiuk, P. Chaignon, and M. Kossakowski, “P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4,” in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 413–421.

[11] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, “The P4→NetFPGA Workflow for Line-Rate Packet Processing,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’19. Association for Computing Machinery, 2019, pp. 1–9. [Online]. Available: https://doi.org/10.1145/3289602.3293924

[12] P. Benácek, V. Puš, and H. Kubátová, “P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers,” in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 148–155.

[13] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, “P4FPGA: A Rapid Prototyping Framework for P4,” in *Proceedings of the Symposium on SDN Research*, ser. SOSR ’17. Association for Computing Machinery, 2017, pp. 122–135. [Online]. Available: https://doi.org/10.1145/3050220.3050234

[14] “Intel® Tofino™Series,” https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html#tofino, [Online; accessed 22-March-2021].

[15] D. Scholz, H. Stubbe, S. Gallenmüller, and G. Carle, “Key Properties of Programmable Data Plane Targets,” in *Teletraffic Congress (ITC32), 32nd International*, 2020, pp. 1–9.

[16] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, “T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors,” in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8.

[17] Y. Zhou, J. Bi, C. Zhang, M. Xu, and J. Wu, “FlexMesh: Flexibly Chaining Network Functions on Programmable Data Planes at Runtime,” in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 73–81.

[18] D. Lukács, G. Pongrácz, and M. Tejfel, “Performance guarantees for P4 through cost analysis,” in *2019 IEEE 15th International Scientific Conference on Informatics*, 2019, pp. 305–310.

A Survey on Domain Impersonation

Derin Amal, Juliane Aulbach* Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: derin.amal@tum.de, aulbach@net.in.tum.de, zirngibl@net.in.tum.de

Abstract—**Domain Impersonation (DI) is a term that collects the different types of attacks that aim to make a user believe that they are communicating with the desired website when they are visiting a maliciously designed phishing website instead. This paper gives an overview of the different categories of domain impersonation, followed by their different types. After looking at the history of domain impersonation, we review countermeasures including browser employed mechanisms, Certificate Authorities (CAs), Certificate Transparency (CT) logs and an automated Framework called ShamFinder that was introduced recently but is not used yet.**

Index Terms—**domain impersonation, IDN homographs, domain name spoofing**

1. Introduction

The term Domain Impersonation (DI) refers to all attacks where attackers with malicious intentions claim to own a domain that is not theirs. With this method, attackers can create a phishing website that imitates a popular website and collect usernames and passwords. This type of attack can have severe consequences if the information phished is sensitive information like passwords for financial accounts and can costs users and the domain owner significant amounts of money. To prevent domain impersonation, one needs to understand the different approaches by users and their characteristics. For this purpose, we will categorize DI and give an overview of different countermeasures that address different types of attacks under those categories and discuss their weaknesses.

2. Background

We differentiate between two types of domain impersonations. First is where an attacker receives a certificate for a domain they do not own and can falsely claim to be someone they are not. The second type is when attackers create their domain and deceive the user by naming their domain similar to the target domain. There are different methods to achieve this, which we will explain in the following. These kinds of domain names are referred to as “spoofing domain names” [1].

It becomes evident that the significant difference is that in the case of the latter, attackers aim to deceive the user directly by abusing humans’ proneness to be inattentive. The first category is characterized by attackers trying to deceive the browser. In a technical matter, browser deception means spoofing the mapping of an IP address to a domain name. Attackers aim to map the IP address of the target domain to their domain name. How this is achieved through different methods will be discussed in Section 3.2. Domain spoofing names, on the other hand, are mapped to their IP address. The attackers’ goal is to make users think that the spoofing domain owned by attackers is actually the domain mapped to the target IP address [2], [3].

To provide the reader with the necessary background information, we will introduce some terms.

CAs are responsible for issuing certificates that bind together a domain name and its public cryptographic key. To verify a CA’s trustworthiness, CAs can issue certificates for other CAs which leads to a CA hierarchy. [4]

Certificate Transparency (CT) logs aim to make the certificate issuance process transparent to the public. CT is an Internet security standard. It keeps a log of certificates issued by trusted CAs to help users identify maliciously issued certificates [2], [5].

Internationalized Domain Name (IDN) allows non-English characters such as Chinese, Cyrillic, and Arabic to be used in domain names and was first proposed by Dürst in 1996 [1]. Currently, IDN is used as an Internet standard [1].

Fully Qualified Domain Name (FQDN) During this paper, we will use the nomenclature proposed by Roberts et al. in which the complete domain name (e.g., google.example.com) is the fully-qualified domain name (FQDN). In our example, google.com would be the domain to be impersonated called the ”target domain” and example.com the actual domain owned by the attackers. [2].

Top Level Domain (TLD) is the last segment of a FQDN, that is, what follows the rightmost dot, e.g., .com in example.com. TLDs can be either generic or country-specific and classify domain names according to their purpose, e.g. .edu for educational facilities or their location, e.g. .de for Germany-based domain names. [6]

3. Domain Impersonation Types and History

In this section, we will give an overview of the different categories of spoofing domain names and browser deception attacks and summarize the transformation of domain impersonation over time.

Type	Example
Typosquatting	facebook.com
Combosquatting	example-site.com could be a spoofing domain name for example.com
Target Embedding	difficult for an average user to differentiate between google.com.site.com and site.google.com
IDN Homograph	éxample.com trying to impersonate example.com

TABLE 1: Examples for different spoofing domain names

3.1. Spoofing Domain Names

There are four common types of spoofing domain names [2]: Target embedding, Unicode Homographs, Typosquatting, and Combosquatting, whereas target embedding is a category recently introduced by Roberts et al. [2].

Typosquatting is the attempt to trick a user by including any 'typos' into a commonly used domain and rely on a user mistaking it for the target domain. Alternatively, attackers who own a Typosquatting domain just hope for users to make a typo while writing a URL in a search bar and access the malicious website by accident [7]–[9].

Target Embedding is a category, where the target domain is embedded in the FQDN in the form of a subdomain. To identify the actual domain, one must read the URL from left to right; the domain name before the TLD is the actual domain. It is important to note that domain impersonation attacks do not include domains that include a target domain owned by the actual domain or cases where the target domain and actual domain are identical. Additionally, Roberts et al. define **subdomain spoofing** as "an umbrella term that includes any attempt at domain impersonation where the target of impersonation is primarily contained in one or more subdomains." [2]. **URL padding** is another form of spoofing, mainly used together with target embedding or combosquatting, where the spoofing domain name is so long that only the deceiving parts(consisting of the target domain) are visible on a user's screen [2].

Combosquatting is similar to target embedding. The target domain is fully included in the FQDN, but contrary to target embedding, the target domain is not a subdomain in this case. [10]

Homographs Unicode Homographs describe the creation of a domain where the name of a commonly used domain is manipulated by using homoglyphs of characters that appear in the target domain. This attack can be highly malicious since some characters are not only confusable but indistinguishable for the human eye and only differ in their Unicode. For the latter, there is no way for a user to detect the attack just by checking the URL.

3.2. Browser Deception Attacks

Cache poisoning in general, is an attack where attackers first request a domain resolution for the target domain and then spoof the response, so the IP address of a domain under their control is cached instead of the respective IP address of the target domain [11]. Another option for cache poisoning to succeed is for attackers to perform a man-in-the-middle attack and eavesdrop on a DNS request. Immediately after, attackers send a spoofed response to the same server. Since they could eavesdrop on the request, attackers know the transaction ID (TxID) entry they use in their spoofed response. This attack exploits the fact that DNS messages are sent without encryption or authentication [12]. Guessing the TxID would be possible, too, but is much less likely to succeed [12]. What makes cache poisoning different from fake certificates is that its essence is to exploit the cache system, which is a memory type. Hence, in case of a successful attack, the consequences will last as long as the false information is stored in the cache. Although it shows to be vulnerable, DNS caching is an essential feature that improves the performance of DNS [12].

Wrongly issued certificates The process of issuing certificates has been proven to be insecure if not carried out correctly [3]. There are different types of validation methods in practice, and they are prone to On-Path attacks. In general, a domain validation process consists of three steps; the application for a certificate, the CA posing a challenge and the applicant doing the challenge, and lastly, the CA checking the challenge and given it is completed, handing out a certificate. There are various ways for a CA to pose a challenge, but we will focus on the abstract process. The key point for an attacker is to fake the successful completion of the challenge, e.g., a DNS challenge where the applicant is supposed to publish a token in the DNS zone file. When the CA checks for completion using a DNS resolver, the attacker spoofs the response tricking the CA into believing that the domain in question is under their control. Note that this is different from cache poisoning since the entries in the DNS resolver cache are not changed. However, the CA is tricked into believing that the IP address of the malicious domain is mapped to the target domain when it is actually not. With a spoofed response, the CA unrightfully issues a certificate to the applicant, which the attacker can use for domain impersonation(e.g., phishing attacks). [13]

3.3. How Did DI Transform over Time?

Although IDN was proposed in 1996 and Gabrilovich and Gontmakher [1] already demonstrated a domain impersonation attack in 2002, homographs were not considered a real threat until IDN started to be widely used around the world with the number being as high as 7.5 million registered IDNs by December 2017 [1]. Also, Hu et al. show that Chrome's defense against IDN homographs that were once 100% effective was not so anymore in their study published two years later [14]. This implies that attackers continue to find new ways to overcome existing security mechanisms. One incident that shows that IDN homograph attacks are a severe issue is the attack on the cryptocurrency exchange company Binance [1]. When companies like Binance are attacked, the consequences for the users and the company can be severe since confidential information will be phished. In the past few years, the possibilities of free certificate issuances like Let's Encrypt have emerged, which had an impact on the number of domain impersonation attacks, too [13]. This is because free issuances give attackers the chance to try attacks without financial barriers [2]. In addition to that, Let's Encrypt uses a fully automated procedure that does not require ownership of domains, but ”it suffices to demonstrate control over the domain's name server” [2].

4. Countermeasures

This section looks at different countermeasures and their effectiveness.

4.1. Browser Employed Mechanisms

There are different mechanisms that browsers use to protect users from malicious phishing websites. Browsers show a lock icon when they could authenticate the website they connected to. However, this is not effective in the case of a spoofing domain name. The lock icon even proves counterproductive since users think that the lock icon ensures the website's "trustworthiness". However, when a user falls for a spoofing domain name attack, e.g., target embedding, they click on a malicious link. The browser authenticates that the user is connected to the URL he requested and shows the lock icon. [2] For the threat of IDN homographs, Browsers have introduced defense policies like once a possible threat is detected, the browser will display the Punycode version of the domain name [14]. Punycode was designed to translate IDN to ASCII compatible encoding, and in this case, it is supposed to warn users of a possibly malicious domain name. Nevertheless, studies have shown that after the browser warns the user with the Punycode mechanism, users are still prone to revisiting the spoofing domain since they are not educated on why their browser uses the Punycode [1]. Furthermore, Hu et al. [14] have shown that all of the browsers they tested (which were the most popular ones) have weaknesses in their mechanisms, and the homographs that bypass those measures are still highly deceiving, continuing to threaten users' data privacy.

4.2. Certificate Authorities

DNSSEC is a layer of security that adds cryptographic signatures to existing DNS records to provide authenticity and data integrity [15]. DNSSEC is one of the most effective options to prevent falsely issued certificates since it protects against both off-path, where attackers do not see the network traffic between the CA and the domain owner's servers but can spoof IP packets by claiming to be the domain owner, and on-path attacks, where attackers can eavesdrop on the network traffic and perform an active man-in-the-middle attack [13]. If the domain is not signed with DNSSEC, several best practices can protect against off-path attacks, e.g., DNS Cookies. Protecting against an on-path attack is not as easy. One solution could be to send redundant queries so that the attacker will not be able to spoof them all. [13] Schwittmann et al. note that "CAs either do not employ all available security measures or fail to implement them properly” [13]. Although DNSSEC is an essential step in fighting cache poisoning and avoiding falsely issued certificates, it has not been widely employed because it adds a layer of complexity [12]. Though DNSSEC is necessary to achieve authenticity and data integrity, it is not entirely secure and has further vulnerabilities that could be exploited. DNSSEC does not provide confidentiality, and it is prone to buffer overruns as well as DDoS attacks. In addition to that, DNSSEC does not tolerate malicious server failures. These are a few of the most critical vulnerabilities pointed out by Ariyapperuma et al.. [15], [16]

4.3. Certificate Transparency

The introduction of CT logs brought many advantages for users as well as domain owners. For instance, domain owners now can easily check for certificates that were issued without their knowledge and hence detect a fake certificate domain impersonation attack, as discussed in Section 3.2, before further harm can happen [17]. In addition to that, since there is a general move towards HTTPS, all sites, including phishing sites, need certificates. Scheitle et al. [17] note that because of that, CT logs can be used to detect phishing domain names They conclude this after a pilot experiment where all valid domain names of a popular company are removed from a list generated from a CT log. As a result, there are many domain names left which partly consist of the companies name and therefore have a high potential of being phishing websites. On the other hand, CT brings with them some risks, too. The transparency allows attackers to scan for unknown domains that would not have been publicly known if it was not for the CT logs [18].

4.4. Preventing Cache Poisoning

Although DNS caching creates the opportunity for cache poisoning attacks, it is an essential feature that improves DNS performance. In case of cache poisoning, security toolbars and phishing filters like Phishtank[1], where users can check if a website was voted to be a phishing website by other users if they suspect it to be one, will not work. Even worse, they will confirm that the domain in question is legitimate since the mapping from the target IP to a malicious domain is cached in the resolver. Since cache poisoning exploits the fact that neither DNS entries nor DNS servers are authenticated, DNSSEC can be used to fight off cache poisoning attacks. It will include an authenticating signature for every valid message. The local DNS server will not accept any responses from attackers who cannot sign their spoofed response. [12]

4.5. ShamFinder

As a response to browsers' insufficient countermeasures against IDN homographs, Suzuki et al. introduce a countermeasure named ShamFinder. ShamFinder is an automated framework to detect IDN homographs. ShamFinder abstractly works as follows: It extracts IDN homographs starting with a database of domain names in the wild. Those domain names are then filtered by the ones starting with the prefix "xn–", implying possible

1. https://www.phishtank.com/

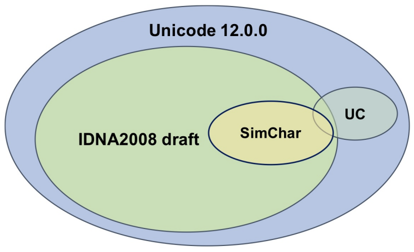


Figure 1: Contamination and overlap of character sets where UC is a recommended mapping for confusable characters [1]

homoglyphs. The next step is to find pairs of the extracted IDNs and popular domain names with the same length (meaning the same number of characters). Note that the extracted IDNs, which possibly are homographs and the list of popular domain names, are two separate sets of data of which the latter can be a ranking list from the Internet. For this purpose Suzuki et al. name Alexa Top Sites[2] as an example. The next step is an algorithm to identify Homographs in the set of extracted IDNs that forms the core of ShamFinder. For the pairs identified in step 2, each character is compared. If two characters at the same index are equivalent, then one proceeds to the next character. If two characters are nonequivalent, then a list of homoglyphs is checked to see whether those characters are homoglyphs. If that is not the case, then the IDN is not considered a homograph. If all characters are equivalent or listed as homoglyphs, the domain will be labeled a homograph. The list of homoglyphs mentioned here is the second contribution of Suzuki et al. [1]. It is named **SimChar** and was constructed as follows: First, each code point is represented as a visual image. Then with a formula as shown in equation 1, the number of different pixels between two glyphs is computed. To summarize, for each pixel, the difference between two images is computed by subtraction. All differences are added up together in the end to obtain a number representing the difference between two glyphs.

$$\Delta = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |I_1(i,j) - I_2(i,j)| \tag{1}$$

Thus a delta equal to zero signifies that both glyphs are visually identical. Now the question is what threshold should be defined to mark the border between homoglyphs and non-homoglyphs. The threshold chosen by Suzuki et al. is 4. A further survey verifies that four is suited as a threshold for what is perceived as confusable by the human eye. The final homoglyph database consists of SimChar combined with what Suzuki et al. call UC, a confusable character database provided by Unicode Technical Standard #39[3].

First of all, the progress made by SimChar in terms of identifying homoglyphs is noteworthy. As seen in Fig 1, SimChar and UC's intersection is relatively small, and it is evident that SimChar has a significant contribution to the number of possible homoglyphs. The major advantage of ShamFinder is that it is an automated framework, meaning it can be expanded whenever new homoglyphs need to be added to the list. The survey done by the authors with human participants verifies that SimChar is a set of glyphs perceived as highly confusing. Therefore one can conclude that the results of ShamFinder will be effective in detecting homographs.

2. https://www.alexa.com/topsites
3. http://unicode.org/reports/tr39/

5. Conclusion and Future Work

To conclude, we can say that DI is a broad topic that brings together vulnerabilities of DNS' different parts. Vulnerabilities in CAs certificate issuance, DNS servers, browsers, and users' behavior can give attackers opportunities to employ DI. One critical suggestions was DNSSEC which is widely known but not implemented by all CAs considered trustworthy. For users, proper education is indispensable and we aim to study the best education approaches in the future. All in all, the issue of DI remains a threat that attackers improve with time, and therefore ever-developing security mechanisms, as well as observation, is necessary. A secure use of IDN can only exist if both sides, user behavior and DNS security, of the problem are approached simultaneously.

References

[1] H. Suzuki, D. Chiba, Y. Yoneya, T. Mori, and S. Goto, "Shamfinder: An automated framework for detecting idn homographs," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 449–462.

[2] R. Roberts, Y. Goldschlag, R. Walter, T. Chung, A. Mislove, and D. Levin, "You are who you appear to be: A longitudinal study of domain impersonation in tls certificates," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2489–2504.

[3] Y. Zeng, T. Zang, Y. Zhang, X. Chen, and Y. Wang, "A comprehensive measurement study of domain-squatting abuse," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[4] C. Crane. (2020) What is a certificate authority (ca) and what do they do? [Online]. Available: https://www.thesslstore.com/blog/what-is-a-certificate-authority-ca-and-what-do-they-do/

[5] C. transparency. How ct works. [Online]. Available: https://certificate.transparency.dev/howctworks/

[6] techopedia. (2021) Top-level domain (tld). [Online]. Available: https://www.techopedia.com/definition/1348/top-level-domain-tld

[7] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Felegyhazi, and C. Kanich, "The long "taile" of typosquatting domain names," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 191–206.

[8] J. Spaulding, S. Upadhyaya, and A. Mohaisen, "The landscape of domain name typosquatting: Techniques and countermeasures," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2016, pp. 284–289.

[9] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis, "Seven months' worth of mistakes: A longitudinal study of typosquatting abuse," in *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS 2015)*. Internet Society, 2015.

[10] P. Kintis, N. Miramirkhani, C. Lever, Y. Chen, R. Romero-Gómez, N. Pitropakis, N. Nikiforakis, and M. Antonakakis, "Hiding in plain sight: A longitudinal study of combosquatting abuse," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 569–586.

[11] J. Trostle, B. Van Besien, and A. Pujari, “Protecting against dns cache poisoning attacks,” in *2010 6th IEEE Workshop on Secure Network Protocols*. IEEE, 2010, pp. 25–30.

[12] R. Bassil, R. Hobeica, W. Itani, C. Ghali, A. Kayssi, and A. Chehab, “Security analysis and solution for thwarting cache poisoning attacks in the domain name system,” in *2012 19th International Conference on Telecommunications (ICT)*. IEEE, 2012, pp. 1–6.

[13] L. Schwittmann, M. Wander, and T. Weis, “Domain impersonation is feasible: A study of ca domain validation vulnerabilities,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 544–559.

[14] H. Hu, S. T. Jan, Y. Wang, and G. Wang, “Assessing browser-level defense against idn-based phishing,” in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[15] cloudfare. How dnssec works. [Online]. Available: https://www.cloudflare.com/de-de/dns/dnssec/how-dnssec-works/

[16] S. Ariyapperuma and C. J. Mitchell, “Security vulnerabilities in dns and dnssec,” in *The Second International Conference on Availability, Reliability and Security (ARES’07)*. IEEE, 2007, pp. 335–342.

[17] O. Gasser, B. Hof, M. Helm, M. Korczynski, R. Holz, and G. Carle, “In log we trust: Revealing poor security practices with certificate transparency logs and internet measurements,” in *International Conference on Passive and Active Network Measurement*. Springer, 2018, pp. 173–185.

[18] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, “The rise of certificate transparency and its implications on the internet ecosystem,” in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 343–349.

Analysis of Wikipedia External Links

Onur Cakmak-Simic, Patrick Sattler, Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: onur.cakmak-simic@tum.de, sattler@net.in.tum.de, zirngibl@net.in.tum.de

Abstract—**Network scans are an inherent element of research within the field of Computer Networks. The basis for these scans is a list of targets, commonly referred to as hitlist. There are readily available hitlists and active research on how such lists can be generated.**

In this paper, we extract domain names from external link datasets provided by the Wikimedia Foundation and use them as the source for generating a hitlist. We assess the general structure of the extracted domains and compare them to the Alexa Top 1M. We find that our list has no apparent structural disadvantages. We also analyze the targets for potential biases regarding their distribution over ASes, prefixes, and IP addresses. Our results show that 52% of the gathered IPv6 addresses are within 30 prefixes of AS13335-CLOUDFLARENET and that the top 10 most occurring ASes contain 45% of all IPv4 targets. We find that 33% of the IPv4 and 42% IPv6 addresses map to more than one domain. Around 5.8% of our domains resolve to the same four IPv4 addresses belonging to AS53831-SQUARESPACE and 3.3% of domains to four IPv6 addresses in AS15169-GOOGLE.

Index Terms—**Internet measurement, Internet hitlists**

1. Introduction

Network scans and their resulting measurements are important to many stakeholders in a network, from individual clients measuring their provided service, ISPs trying to optimize their operational costs to researchers measuring network characteristics, evaluating their findings, or deploying algorithms on a larger scale. IPv4 scanning and the generation of hitlists date back to the 90s [1,2]. Nowadays, tools like ZMap [3] and MASSCAN [4] enable scanning the entire IPv4 address space in feasible time. Although possible, a full scan might not be suitable. Not all types of scans scale well to that size [5]. We might need domains names, e.g., for TLS scans which generally require domain names due to Server Name Indication, we might have limited infrastructure or have a narrow target group. With IPv6, complete scans of the address space are not feasible [6], so hitlists are a necessity.

Depending on the source, the list of targets might be biased. Detecting and eliminating these biases is not trivial and a field of active research [7,8]. To ensure some form of quality for the list of targets, Gasser et al. [7] suggest gathering addresses that belong to individual hosts and have an even distribution across ASes and prefixes. At the very least, the potential for biases should be considered when conducting research, as a nonrepresentative or skewed list might lead to wrong conclusions.

In this paper, we analyze a potential source for generating a hitlist. For that we extract domain names from external links found in Wikipedia articles. According to Wikipedia community guidelines [9], each article may include an external link section listing the web presence of entities relevant to the article. External links refer to links from articles to web pages outside of Wikipedia.

Outline Section 2 briefly presents related work and terminology used throughout this paper. Our methodology for extracting the domains and resolving them to IP addresses is outlined in Section 3. Section 4 covers some structural properties of the extracted domain names. We inspect the list of targets for potential biases towards ASes, prefixes, and IP addresses in Section 5. Finally, Section 6 concludes our paper and suggests possible future work.

2. Related Work and Background

There are many sources from which to generate hitlists, including passive [6] and active measurements [10], Certificate Transparency logs [11], and machine learning [12]. As a result of continuous research, a considerable amount of datasets, providing sources or targets, have been accumulated. While some of these datasets are restricted and proprietary [6,13], many are publicly available [14,15]. Frequently used sources are top lists, e.g., the Alexa Top 1 Million list [16] that rank web domains by popularity. Scheitle et al. [17] and Pochat et al. [18] found that some of these lists exhibit characteristics that need to be accounted for prior to their use in research. These include, but are not limited to, significant and frequent churn, a nontransparent ranking mechanism, and a weekend and clustering effect [5]. Attempts to address some of these issues include using prefix top lists [19] or incorporating multiple such lists [20].

To the best of our knowledge, Paul Hoffman's [21] work is the first to generate a hitlist using external links from Wikipedia articles and to evaluate specific network characteristics of the targets.

Background For the structural analysis in Section 4, we use the notions of base domain and subdomain depth to obtain insights into the depth and breadth of our domains [17]. By base domain, we refer to the public suffix and the first domain prefixing it, e.g., `google.com`. Each subdomain preceding the base domain adds a value of 1 to the subdomain depth, e.g., `www.support.google.com` has subdomain depth 2. For clarity, in this paper, the term bias

TABLE 1: List structures. The SD_x columns indicate the share of domains with subdomain depth x. SD_0 represents domains with no subdomain, i.e., a base domain [17]. In the third column, 1498 TLDs correspond to 100%. The given numbers are rounded down to the nearest tenth decimal.

List	Size	TLDs	SD_0	SD_1	SD_2	$SD_{>3}$	$\cap$Alexa
Joint	3.5M	57.8%	22.2%	70.3%	6.2%	0.8%	21.0%
de	1.2M	44.1%	17.7%	76.2%	5.7%	0.5%	8.5%
en	3.3M	57.6%	23.9%	68.7%	6.5%	0.7%	20.9%
fr	981.8K	41.7%	17.9%	74.2%	6.8%	0.8%	8.5%
ceb	6.2K	9.9%	17.6%	76.9%	5.1%	0.2%	0.3%
sv	243.7K	28.1%	16.5%	75.5%	6.3%	1.5%	3.4%
nl	317.3K	31.2%	14.0%	78.9%	5.3%	1.6%	3.6%
Alexa	690.9K	52.2%	85.4%	14.4%	0.1%	0.0%	100.0%

of a hitlist refers to its propensity to certain subsets in the IP address space.

3. Methodology

The Wikimedia Foundation, the parent company of Wikipedia, provides a wide range of data dumps. Among them are SQL dumps that provide information about the external links across all articles within a given Wikipedia language edition. As of June 2021, there are 321 Wikipedia editions. For this paper, we used the six largest editions, based on the number of articles. These are the English, Cebuano, Swedish, German, French, and Dutch Wikipedias. To create the lists of domain names we performed the following steps:

- We pulled the external link SQL dump for each language edition on May 04, 2021.
- We extracted the individual URLs from the dumps and pruned those that were nonvalid URLs, had bad syntax, used nonstandard ports, or contained irrelevant protocols.
- We removed any unwanted prefix and suffix leaving the base domain and subdomains.
- We deleted duplicate entries.

In addition, we created a Joint list by merging the individual lists, again removing duplicate entries. To resolve the domains and collect IP addresses, we used MassDNS [22] with an Unbound [23] resolver. We performed the scan on May 17, 2021.

4. Structure

We check how many unique Top Level Domains (TLDs) are used and the subdomain depth across the domains in each list. In addition, we compute the intersection between our lists and the Alexa Top 1M, which we retrieved May 26, 2021.

4.1. TLD Coverage

As of May 2021, IANA [24] reports the existence of 1498 valid TLDs. Table 1 shows the results for all lists. The Joint and English list with 57.8% and 57.6% cover almost the same number of TLDs, approximately 865. There is a noticeable relation between the size of a list and the amount of TLDs it contains. An exception is the Alexa Top list which at almost half the size of the German list includes 120 TLDs more. This might be attributed to the larger share of base domains in the Alexa list leading to a wider range of targets. The smallest list, Cebuano, misses over 90% of TLDs. This is a consequence of its small number of entries, although it is the second largest Wikipedia edition. It is the smallest list because of an unexpectedly large number of duplicate entries in the SQL dump, which we removed during the list's creation.

TABLE 2: Top 5 TLDs by occurrence. Values are percentages of the number of domains in the respective list.

	Lists					
TLD	de	en	fr	ceb	sv	nl
com	25.4	48.9	38.2	44.5	32.0	26.6
org	7.7	14.2	11.2	11.9	9.3	7.6
de	32.4	2.4	3.6	1.5	6.4	6.8
net	3.3	4.2	4.3	4.0	3.5	3.1
fr	1.6	0.7	12.2	2.9	0.8	1.5

Table 2 lists the five most frequently occurring TLDs across the language-specific lists. Three of the most commonly used TLDs on the internet, `com`, `net`, and `org` are present. The entries `de` and `fr` are due to an unsurprising bias of the German and French list, the second and third largest lists respectively, towards these TLDs. Around 420 K domains in the German list have `de` as their TLD and around 118 K entries in the French list have TLD `fr`.

We would like to note that the extracted URLs contained thousands of invalid TLDs, which was a point of interest in previous research [17] when analyzing such lists. Due to the human component in adding external links to articles, this is to be expected and not further elaborated on in this paper.

4.2. Subdomain Depth

Looking at the subdomain depths in Table 1, we notice a significant discrepancy between the Wikipedia lists and the Alexa Top list. With 590 K entries, the Alexa list almost exclusively consists of base domains, whereas our lists comprise around 14% to 23% base domains each. Conversely, up to 78% percent of domains in the Wikipedia lists have subdomain depth 1, compared to Alexa's 14%. Worth noting is that 60-70% of these domains with subdomain depth 1 have the `www.` prefix, which

TABLE 3: Top 10 ASes by the number of contained domains from the Joint list.

IPv4				IPv6			
AS	Domains	Addresses	Prefixes	AS	Domains	Addresses	Prefixes
AS13335 - CLOUDFLARENET	442K	80K	175	AS13335 - CLOUDFLARENET	399K	75K	30
AS16509 - AMAZON-02	209K	49K	1.5K	AS6724 - STRATO	49K	473	2
AS53831 - SQUARESPACE	208K	27	3	AS8560 - IONOS-AS	48K	2.8K	3
AS15169 - GOOGLE	174K	25K	266	AS16509 - AMAZON-02	29K	15K	143
AS58182 - wix_com	147K	25	4	AS8972 - Host Europe	21K	2.8K	2
AS14618 - AMAZON-AES	140K	24K	108	AS51468 - ONECOM	20K	20K	1
AS16276 - OVH	137K	37K	87	AS15169 - GOOGLE	20K	222	14
AS8560 - IONOS-AS	118K	10K	34	AS20773 - GODADDY	13K	10K	1
AS46606 - UNIFIEDLAYER-AS-1	90K	27K	129	AS16276 - OVH	12K	1.9K	5
AS26496 - GO-DADDY-COM	26K	9K	285	AS54113 - Fastly	10K	219	16

does not provide us with any more interesting targets than base domains. Our lists do contain a considerable amount of domains with subdomain depth 2 or greater, leading to potentially interesting targets. In the Joint list, there are ≈210 K domains with subdomain depth 2 and ≈28 K with a subdomain depth larger than 3. The Alexa Top list has 740 and 35 such domains, respectively. This suggests that our Joint list covers domains beyond an entity's main web presence.

4.3. Intersection with the Alexa Top 1M

The intersection between hitlists is an important measure and was studied in previous research [17] as a large overlap may indicate that potential biases and shortcomings in one list are also present in the other. Of the approximately 3.5 M domains in our Joint list, about 150 K can be found on the Alexa Top list. Most of this overlap comes from entries in the English list. All other lists have intersections consistently below 10% and in total only contribute 7K domains to the overlap of the Joint list. This indicates that our lists are a more diverse source for the generation of a hitlist that goes beyond the most popular domains. The generally low overlap might be explained by the fact that the broad diversity of Wikipedia articles results in many external links pointing to niche, regional and unknown domains.

5. Biases

In this section, we analyze our target address for potential biases by inspecting their distribution over ASes, prefixes, and IP addresses. We conclude the section by checking IPv6 adoption and the use of privacy extensions across our targets. The following analysis is based on the addresses resolved from the Joint list only.

5.1. AS and Prefix Distribution

Table 3 shows the top 10 ASes most domains within our list belong to. For both IPv4 and IPv6, CLOUDFLARENET is in first place. About 11% of all domains resolved to an IPv4 address and 52% of IPv6 addresses resolved to are within AS13335-CLOUDFLARENET and 175 and 30 of its prefixes respectively. Six of the 10 ASes are found on both sides, while SQUARESPACE, wix_com, AMAZON-AES, and UNIFIEDLAYER-AS drop out of the top 10 when considering IPv6 addresses. With STRATO, IONOS-AS, Host Europe, and GODADDY, about 130 K (17.1%) of all IPv6 domains are located in a German AS and within 8 of their prefixes.

The domains are distributed over 19976 ASes (IPv4) and 2304 ASes (IPv6), yet the top 10 ASes contain 45% and 80% of them respectively. These results carry over to the approximately 69 K IPv4 and 3.4K IPv6 prefixes covered in total. All domains in the top 10 ASes are within 2591 (3.7%) and 217 (6.2%) of all covered prefixes. Figure 1 provides a graphical representation of these results. Beyond the top 10, we find that the top 100 ASes cover 77% and the top 250 approximately 85% of all IPv4 domains. For IPv6, it is more significant as the respective number of top ASes contain 96% and 97% of all domains.

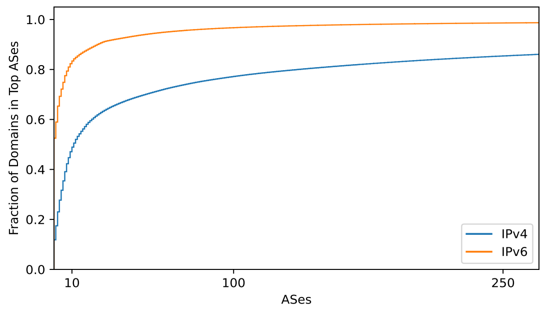


Figure 1: CDF showing address distribution over top ASes.

5.2. IP Addresses

To identify a possible bias towards a small set of IP addresses, we check how many domains are resolved to the same address. In total, we gathered 887 K unique IPv4 and 175 K unique IPv6 addresses.

IPv4 Table 4 shows the top IPv4 addresses appearing the most in our hitlist. Given the purpose of external links, it is no surprise that most addresses belong to well-known web hosters like SQAURESPACE and Wix.com. There are no significant differences among the Top 8, with approximately 50 K occurrences each. In total, around 450 K (12.8%) domains are resolved to these addresses. Figure 2 shows the distribution over the top 5000 IPv4 addresses. We see that the top 100 addresses account for 22%, the top 1000 for 32%, and the top 4000 for 39% of all domains. After the top ≈300 K addresses, we have a one-to-one mapping between domain and address.

IPv6 In Table 5, the top 10 most occurring IPv6 addresses are listed. Structurally, the table is similar to that of the IPv4 addresses. The top 5 addresses are resolved to from

TABLE 4: Top 10 most frequently occurring IPv4 addresses in the target list.

Address	#	AS
198.185.159.144	53 K	SQUARESPACE
198.185.159.145	50 K	SQUARESPACE
198.49.23.145	50 K	SQUARESPACE
198.49.23.144	50 K	SQUARESPACE
185.230.63.171	49 K	wix_com
185.230.63.107	49 K	wix_com
185.230.63.186	49 K	wix_com
184.168.131.241	47 K	GO-DADDY-COM-LLC
3.223.115.185	29 K	AMAZON-AES
192.0.78.24	23 K	AUTOMATTIC

around 4800 domains each where the top 4 belong to AS15169-GOOGLE. In total, the top 10 addresses cover around 39 K (5.2%) of all domains. Looking again at Figure 2, we find that the distribution over IPv6 addresses follows a similar slope to that of the IPv4 addresses.

TABLE 5: Top 10 most frequently occurring IPv6 addresses in the target list.

Address	#	AS
2001:4860:4802:32::15	4886	GOOGLE
2001:4860:4802:36::15	4842	GOOGLE
2001:4860:4802:34::15	4840	GOOGLE
2001:4860:4802:38::15	4837	GOOGLE
2a05:d014:9da:8c10:306e:3e07:a16f:a552	4650	AMAZON-02
2a01:238:20a:202:1086::	3599	STRATO
2a01:238:20a:202:1162::	3218	STRATO
2003:2:2:15:80:150:6:143	2840	DTAG
2606:4700:90:0:b518:199c:8a1f:d33b	2736	CLOUDFLARENET
2a01:238:20a:202:1064::	2393	STRATO

The top 100, 1000, and 4000 IPv6 addresses account for 14%, 25%, and 29% of all domains respectively. Here we have a one-to-one mapping between domain and address after 74 K addresses. Interestingly another German AS, DTAG, is the only one appearing in either address table, which is not part of Table 3.

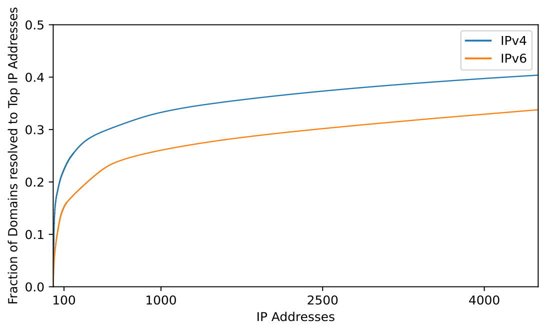


Figure 2: CDF showing domain distribution over top IP addresses.

IPv6 adoption IPv6 adoption across the internet was a network characteristic of interest in previous research [25]. Of our ≈3.5 M domains, around 750 K could be resolved to an IPv6 address. This represents an adoption of 21.7%. We take the native IPv6 traffic google receives [26] as a reference for the adoption on the internet, which is 31% as of June 02, 2021. Our list falls well below that. This again might be because of the diverse, possibly niche, and regional nature of external links. Additionally, we determine how many domains can be resolved to an IPv6 address, whereas Google passively measures user traffic.

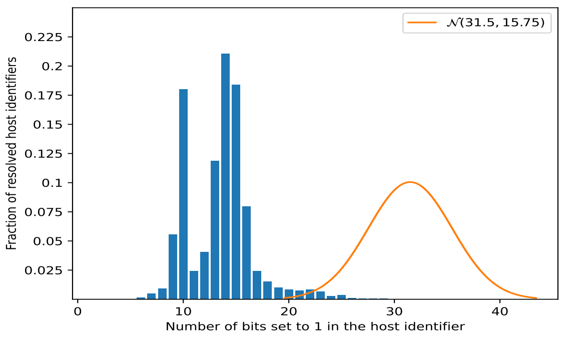


Figure 3: Bit distribution over IPv6 host identifiers.

Privacy Extensions RFC 4941 introduced privacy extensions to reduce the traceability of MAC addresses due to the use of Stateless Address Autoconfiguration. Using privacy extensions, the interface identifier, i.e., the last 64 bits, are replaced by random bits. An approximation for the sum of these single bit distributions is the normal distribution $\mathcal{N}(31.5, 15.75)$ [6]. We analyzed the interface identifiers of our IPv6 addresses. The distribution for the sum over the bits is shown in Figure 3. We see that our sample of host identifiers does not match the normal distribution. This shows that most of our targets are not using privacy extensions. Considering that most of our targets are presumably web servers having no need to mask their host identifiers for the sake of reducing traceability, this is not too surprising.

6. Conclusion and Future Work

In this work, we analyzed external links from Wikipedia articles as a source for creating a hitlist. We found that our Joint list has similar TLD coverage and higher average subdomain depth than the Alexa Top list. Around 21% of the domains in the Alexa list are also present in the Joint list. When evaluated for biases, our hitlist showed a significant propensity towards a small number of ASes and prefixes. In addition, a large portion of domains are resolved to a small set of IPv4 and IPv6 addresses. We have seen that the IPv6 adoption of our targets is below the general adoption and that most of them do not use privacy extensions.

We note that this work evaluated the hitlist in isolation without comparing it with existing alternatives. This could be addressed in future work to determine the relative value of this method. An attempt to eliminate found biases might increase the quality of the hitlist. Other potential aspects for future work include assessing possibilities to manipulate external links, monitoring the change of the domain names over a longer period of time, considering further network characteristics, checking for additional biases, and incorporating additional Wikipedia language editions.

References

[1] J.-J. Pansiot and D. Grad, "On Routes and Multicast Trees in the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 1, p. 41–50, Jan. 1998.

[2] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, 2000, pp. 1371–1380 vol.3.

[3] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 605–620.

[4] R. Graham, "MASSCAN: Mass IP port scanner," Available at https://github.com/robertdavidgraham/masscan, [Online; accessed 01-June-2021].

[5] W. Rweyemamu, T. Lauinger, C. Wilson, W. Robertson, and E. Kirda, "Clustering and the Weekend Effect: Recommendations for the Use of Top Domain Lists in Security Research," in *Passive and Active Measurement*, D. Choffnes and M. Barcellos, Eds. Cham: Springer International Publishing, 2019, pp. 161–177.

[6] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle, "Scanning the IPv6 Internet: Towards a Comprehensive Hitlist," in *In Proceedings of the Traffic Monitoring and Analysis Workshop*, 2016.

[7] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczyński, S. D. Strowes, L. Hendriks, and G. Carle, "Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 364–378.

[8] A. Murdock, F. Li, P. Bramsen, Z. Durumeric, and V. Paxson, "Target Generation for Internet-Wide IPv6 Scanning," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 242–253.

[9] Wikipedia, "Wikipedia:External links," Available at https://en.wikipedia.org/wiki/Wikipedia:External_links, 2021, [Online; accessed 24-July-2021].

[10] P. van Dijk, "Finding v6 hosts by efficiently mapping ip6.arpa," Available at https://web.archive.org/web/20161121215042/http://7bits.nl/blog/posts/finding-v6-hosts-by-efficiently-mapping-ip6-arpa, [Online; accessed 01-June-2021].

[11] F. Marquardt and C. Schmidt, "Don't Stop at the Top: Using Certificate Transparency Logs to Extend Domain Lists for Web Security Studies," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 409–412.

[12] P. Foremski, D. Plonka, and A. Berger, "Entropy/IP: Uncovering Structure in IPv6 Addresses," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 167–181.

[13] D. Plonka and A. Berger, "Temporal and Spatial Classification of Active IPv6 Addresses," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 509–522.

[14] R. P. Sonar, "Forwards DNS Data," Available at https://opendata.rapid7.com/sonar.fdns_v2/, [Online; accessed 27-May-2021].

[15] R. NCC, "IPMap," Available at https://ftp.ripe.net/ripe/ipmap/, [Online; accessed 27-May-2021].

[16] Alexa, "Top 1M sites," Available at https://www.alexa.com/topsites, [Online; accessed 18-May-2021] http://s3.dualstack.us-east-1.amazonaws.com/alexa-static/top-1m.csv.zip.

[17] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. D. Strowes, and N. Vallina-Rodriguez, "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 478–493.

[18] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, 2019.

[19] J. Naab, P. Sattler, J. Jelten, O. Gasser, and G. Carle, "Prefix top lists: Gaining insights with prefixes from domain-based top lists on dns deployment," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 351–357.

[20] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A Lustrum of Malware Network Communication: Evolution and Insights," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 788–804.

[21] P. Hoffman, "Collecting Typical Domain Names for Web Servers," Available at https://www.icann.org/en/system/files/files/octo-023-24feb21-en.pdf, 2021, [Online; accessed 02-May-2021].

[22] T. U. of Munich, "MassDNS," Available at https://github.com/blechschmidt/massdns, 2021, [Online; accessed 09-June-2021].

[23] N. Labs, "Unbound," Available at https://github.com/NLnetLabs/unbound, 2021, [Online; accessed 09-June-2021].

[24] IANA, "TLD Directory," Available at https://data.iana.org/TLD/tlds-alpha-by-domain.txt, 2021, [Online; accessed 01-June-2021].

[25] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, "Measuring IPv6 Adoption," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 87–98.

[26] Google, "IPv6 Statistics," Available at https://www.google.com/intl/en/ipv6/statistics.html, 2021, [Online; accessed 01-June-2021].

Survey on SR-IOV performance

Maximilian Fischer, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: maximilian.fischer@in.tum.de, wiedner@net.in.tum.de

Abstract—**Scalable, high performance VM networking is becoming increasingly important. Paravirtualized solutions like VIRTIO are not up to the task, since the overhead in latency and bandwidth is too high. Single-Root I/O Virtualization (SR-IOV) is a technology which eliminates the need to emulate NICs and could exceed VIRTIO and similar solutions in terms of performance. In this paper we give an overview over the performance of SR-IOV with ethernet, focusing on latency since it is especially important for applications like network function virtualization. We look at the current status of SR-IOV, as well as some optimizations that can be applied and how they actually impact performance and particularly latency. We discover that latency has not been the focus of recent research, but rather bandwidth. Additionally, the scalability of stock SR-IOV and of the shown optimizations is not examined enough, especially with regard to latency. We come to the conclusion that further research is necessary.**

Index Terms—**SR-IOV, network function virtualzation, measurement, hight-speed networks, ethernet networks**

1. Introduction

As more and more services become virtualized and containerized, high performance networking becomes increasingly important. An application where this is especially critical are virtual network functions (VNF). Here the network functions which would normally be handled by separate devices, like a firewall, router or DNS resolver are instead put into virtual machines (VMs), with usually no specialized hardware. These are applications which all other clients in the network depend on, since, depending on the VNF, either traffic goes through the VM, e.g. a router, or future traffic depends on it, e.g. a DNS resolver. Low latency and high bandwidth are a must, since VNFs can act as a network-wide bottleneck. Since multiple VNFs can run on a single host, good scalability is also paramount, otherwise a major advantage of VNFs is lost.

At the moment, VM networking is implemented with different systems, depending on the hypervisor used. When using the Kernel-based Virtual Machine (KVM), a very popular paravirtualization standard is VIRTIO. The hypervisor Xen also has capabilities to use paravitualized network interfaces [1]. Unfortunately, neither KVM nor Xen can provide performance near that of native networking. Bandwidth is severely limited by the number of interrupts the CPU can handle. Latency is impacted negatively by tx batching, a technique where the hypervisor doesn't inform the VM about every packet that arrives, but rather does so in batches, thus reducing the number of context changes. When turning off tx batching, latency benefits but bandwidth suffers due to the number of interrupts [2] [3]. A technology which could solve all of this is Single Root I/O Virtualization (SR-IOV). An SR-IOV capable network interface controller (NIC) can present itself as multiple virtual PCIe devices. These devices are split into virtual functions (VF) and one physical function (PF). The VFs are passed through to the VMs and the PF is for the host. The PF has the capability to configure the NIC, how many VFs it has, the routing and much more, depending on the NIC [2]. In theory, this greatly improves performance, since the hypervisor doesn't have to emulate or paravitualize the NIC. There are a lot of works discussing the praxis, unfortunately most of them focus on bandwidth. If latency is ever talked about it is mostly in the context of InfiniBand. That is because latency is very important for most applications using InfiniBand, like the message passing interface (MPI) [4].

The goal of this paper is to give an overview of the current state of the performance of SR-IOV networking when using ethernet, with a focus on latency. First we will talk about the current, unoptimized state of SR-IOV. Afterwards we will show some optimizations which can be applied and how they actually impact the performance. Lastly we will show what conclusions we can draw from this.

2. Current Status

There are several works analysing and discussing the performance of SR-IOV as is, without any optimizations. In this section we take a look at some of them.

A general performance overview is given by Liu [2] with a 10 GbE connection between two servers. The half-roundtrip latency is shown in Figure 1. The 7 µs difference between SR-IOV and native is attributed to the interrupt virtualization needed for SR-IOV. The very high latency of VIRTIO can be traced back in part to tx batching. In this approach the hypervisor, while writing incoming packets into the buffer of the VM, does not instantly inform the VM about them. Instead the VM is only interrupted every few packets, which leads to a dramatically lower number of interrupts. When disabling the tx batching of the VIRTIO network interface, latency improves it to around 37 µs while negatively impacting performance under high tx load due to more interrupts. Figure 2 shows the bandwidth compared to the CPU usage when receving. For large messages, the performance of SR-IOV and native is almost

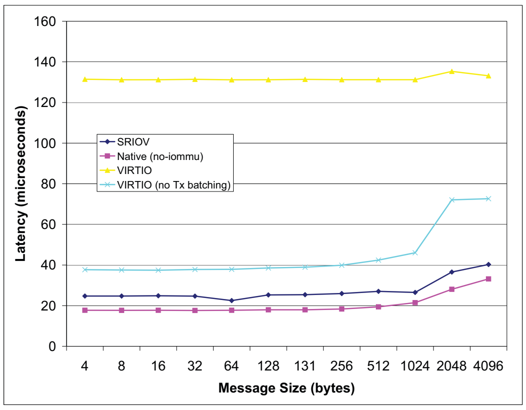


Figure 1: Latency [2]

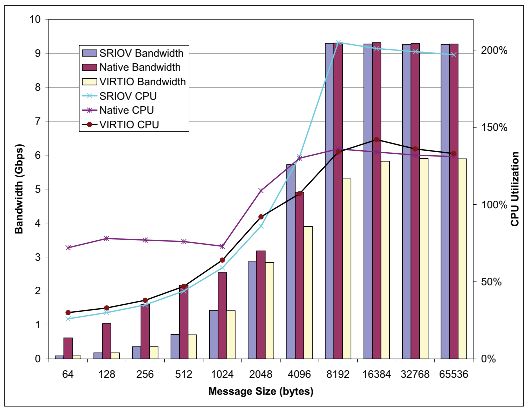


Figure 2: Rx bandwidth compared against CPU usage [2]

the same, at around 9.1 Gbit/s, while VIRTIO reaches its maximum at 5.9 Gbit/s, with a theoretical maximum of 10 Gbit/s. The CPU usage for SR-IOV, however, is significantly larger for messages bigger than 4096 B, at 200 % compared to 140 % for native or VIRTIO. This is important to note, since the CPU is often the bottleneck when using SR-IOV and also the point where a lot of papers discussed later begin their optimizations. The performance when receiving translates more or less to transmitting, except that native and SR-IOV already start to diverge for messages smaller than 8192 B. Also, the CPU usage of 110 % is closer to the VIRTIO CPU usage of 140 % than to the native of 60 %. A category in which VIRTIO pulls ahead of SR-IOV is VM exits and interrupt requests. The number of SR-IOV VM exits exceeds that of VIRTIOs 18000 by nearly eightfold [2].

Lockwood et al. [4] analyse the MPI performance of SR-IOV 10 Gbit/s ethernet on a VM compared to that of an VM with network virtualization, native 10 Gbit/s ethernet, SR-IOV InfiniBand on a VM and native InfiniBand. Since we focus on ethernet in this paper, InfiniBand is not discussed. The tests using 10 Gbit/s ethernet on a VM are carried out using Amazon Web Services and with no other VM running on the host. The benchmarks reveal that, as expected, native performs better than SR-IOV, which in turn performs better than fully virtualized networking. Latency stays about the same for message sizes of ≤ 1 KiB, with SR-IOV having about 40 % lower latency than fully virtualized networking. In numbers this means that fully virtualized networking has 65 µs of latency while SR-IOV has 40 µs. Unfortunately, this is still about two times as high as native. For larger messages the difference in ping between SR-IOV and fully virtualized starts to decrease, while the absolute latency of all three variants gets bigger. At the largest tested message size of 4 MB SR-IOV still performs 12 % better than without SR-IOV. SR-IOV also performs better when looking at latency variation, due to the fully virtualized networking being influenced by other tasks on the CPU. In numbers, that means SR-IOV has three to four times less latency variation. The bandwidth analysis shows that native provides nearly 6.4 Gbit/s for unidirectional and the full 10 Gbit/s for bidirectional traffic. The bandwidth of SR-IOV and fully virtualized never gets above about 3.2 Gbit/s for unidirectional, and 4 Gbit/s for bidirectional traffic. These differences to the previous paper are likely to the difference in benchmarks being used, as well as the usage of MPI in this paper [4].

The performance of SR-IOV is compared against that of Open vSwitch Data Plan Development Kit (OVS-DPDK) and Fast Data input/output Vector Packet Processing (FD.io VPP) for use with VNFs by Pitaev et al. [5]. The theoretical maximum interface speed is 20 Gbit/s. When the VNFs are under a light load, just doing IPv4 Forwarding, SR-IOV clearly pulls ahead. While the throughput of SR-IOV scales nearly linearly with every additional VNF added, up to about 19 Gbit/s, the throughput of FD.io VPP and OVS-DPDK stops increasing at two and three VNFs respectively, at about 12 Gbit/s to 16 Gbit/s. When using packet sizes of 128 B instead of the previous IMIX, the performance difference becomes even clearer, though the general development is the same. Loading the VNFs with NAT, Firewall, QoS and DPI and IMIX or 1500 B packet sizes yields much more interesting results. SR-IOV scales nearly linearly to the maximum bandwidth with both IMIX and 1500 B packets, while OVS-DPDK does not reach it at all and FD.io VPP only for 1500 B packets [5].

Xu and Davda [6] talk about SRVM, which provides VM live migration support for SR-IOV to the VMware ESXi hypervisor. Comparing the performance of SR-IOV to that of the VMXNET3 driver, SR-IOV performs, as expected, better. All the tests were conducted with a 10 GbE NIC. The average latency of the system normalized to the native latency is 113 % for SR-IOV and 207.7 % for VMXNET3. The minimum and maximum latency of SR-IOV is even better than native, at 95.2 % and 66.2 %, while that of VMXNET3 is 185 % and 636.7 % respectively. As expected, the throughput of SR-IOV is nearly on par with native at 99.8 % for packet sizes of 256 B, 512 B and 1024 B, whereas VMXNET3 is at 16.2 %, 33.5 % and 49.7 % [6].

Hwang et al. [7] present NetVM, a tool which competes with SR-IOV and makes use of the data plane development kit and KVM. It consists of three parts, NetVM manager, NetVM core engine and NetLib. NetVM manager is the interface to the hypervisor, receiving events from it. It then notifies NetVM core engine which actually implements those events. NetVM core engine is also responsible for receiving packets and forwards it to the VM via shared memory over an emulated PCI device. NetLib provides the interface for the user application in the VM. The test setup consists of two servers with a 10 GbE connection between them. When comparing the roundtrip latency of NetVM and SR-IOV while forwarding packets, both behave mostly the same for lower latencies, at about 40 µs to 50 µs. But the latency of SR-IOV starts to rise shortly above 5 Gbit/s of load, to 70 µs, while NetVM stays more or less the same. Although not mentioned explicitly, this is probably due to the limitations of SR-IOV seen in other papers discussed previously. Since here SR-IOV is not optimized in any way, it presumably starts to reach the limits of what the CPU can handle interrupt-wise. Unfortunately CPU load is not measured here [7].

Bauer et al. [8] compare the performance of SR-IOV software function chaining (SFC) using a single PF to that of SFC using VFs. When just comparing the performance of the ixgbe and the ixgbevf drivers, latency over an Open vSwitch OvS bridge is also examined. For bandwidths $\leq$500 Mbit/s both are around 10^4 µs. Above that, ixgbe is at about 0.8×10^7 µs, while ixgbevf is at 0.8×10^6 µs. This is presumably due to the tasks which would normally be performed by the OS now being performed by the NIC directly, which is especially beneficial for SFCs, since the forwarding between two SFs can be offloaded onto the NIC. When looking at interrupts, VF and PF mostly behave the same for loads below 700 Mbit/s. Above that, the number of interrupts for PF decreases drastically, from 10^4 Interrupts/s to 10^2 Interrupts/s, while the interrupt numbers for VF stay mostly the same at 10^4 Interrupts/s. As already seen previously, this is due to the lack of a number of features concerning the controlling of interrupts, like dynamic interrupt throttle rate. When comparing the scalability of SCFs using SR-IOV to that of SFCs using virtual ethernet (vEth) interfaces, the results show that using vEth yields higher throughput for the tested chain lengths from 1 to 5 SFs. This is explained with VFs having less efficient drivers and that there are more I/O operations required. Examining the throughput chains of length one, two and four with varying load, vEth performs better initially. For chains with length two and four, SR-IOV still takes back the lead after a certain threshold. Due to the software nature of vEth, as soon as all resources are taken up, the whole system suffers. Since with SR-IOV, a big part of the processing is offloaded onto the NIC, this is not a problem [8].

3. Optimizations

There are several different optimization options. A lot of them focus on decreasing the CPU load on high throughput, though there are of course others. In this section we discuss several of them.

Dong et al. [3] explore the performance of SR-IOV together with the hypervisor Xen. Three major optimisation methods are proposed and tested, all concentrating on interrupts and the minimisation of performance hits from them. One of them being the moving of message signalled interrupts (MSI) from the device model in userspace into the hypervisor. The emulation of a virtual end of interrupt (EOI) is also identified as a hotspot and simplified massively. Thirdly, an adaptive interrupt coalescing (AIC) method is developed, which adjusts the interrupt rate dynamically based on a set of equations to reduce load on the CPU. This optimization presumably has the most impact on latency, though unfortunately this is not measured. A 9.6 % drop in TCP throughput is observed when using an interrupt frequency of 1 kHz, which is attributed to TCPs latency sensitivity, though it is mitigated by the AIC optimisation. Since the bandwidth is at its maximum of around 1 Gbit/s per guest even before the optimizations, it is barely affected by them. As is expected, overhead of the CPU is heavily reduced, together the optimizations reduce it from 499 % to around 227 %. This mostly affects the host, but some improvements are also made in the CPU overhead of the guests. When testing the scalability of the three optimizations no major issues are detected, even at 60 VMs, although it might be important to mention that there are still only a total of 10 1 Gbit/s connections available [3].

Li et al. [9] explore the method of throttling the frequency of these interrupts on the fly to increase the throughput. The method of using a fixed interrupt rate (FIR) implemented in some device drivers is used as a baseline for measurements with the baseline fixed at 8000 Interrupts/s. Two new approaches of regulating interrupts are proposed. Course grained interrupt rate control (CGR) classifies the traffic into four categories, depending on the packet size. Based on that, the interrupt rate is set, higher for smaller packets and lower for larger packets. Packets with sizes from 64 B to 300 B are classified as latency sensitive traffic and the interrupt rate is set to 20 kInterrupts/s. Packets smaller than 64 B are classified as latency critical traffic, the interrupt rate is set to 100 kInterrupts/s. How this actually translates to praxis is not measured. The idea behind Adaptive interrupt rate control (AIR) is that there is always an optimal interrupt rate, depending on the currently used bandwidth and average packet size, which can be calculated. It is set based on the current bandwidth, average packet size, number of packets received on each interrupt and the current interrupt rate. Comparing these three approaches using netperf TCP and UDP stream, it becomes clear that for 4 VMs or less the CPU overhead is large enough that the CPU performance is worse using AIR and CGR compared to FIR. When using TCP stream, a packet size of 1472 B and four VMs, the CPU usage for AIR, CGR and FIR is ~500 %, ~400 % and ~395 % respectively. That said, the throughput when using AIR and CGR is consistently on par or higher than when using FIR, for higher VM numbers higher than 8 AIR even outperforms CGR [9].

Huang et al. [10] talk about optimizing SR-IOV with the AIR also discussed by Li et al. [9] as well as an approach using multi-threaded NAPI. New API (NAPI) is an API in the Linux kernel allowing the driver using it to mask some of the interrupts produced by incoming packets. It normally is single-threaded, which makes it more and more of at bottleneck with increasing VM counts, since they all are limited by the capacity of this single thread [11]. A multi-threaded NAPI is proposed, consisting of a dispatcher and n worker threads. This increases the throughput but also the CPU usage. For example when using TCP with a packet size of 1472 B, throughput increased by 38 %, from ~5.8 Gbit/s to ~8 Gbit/s, but CPU usage also increased by 73 %, from ~100 % to ~175 %. For smaller packets, the performance gain is barely noticeable, it is up to about 0.6 Gbit/s from 0.5 Gbit/s. The increase in CPU usage is similarly small, from 60 % to 65 %. UDP performance is behaves similar, but it is lower in general, due to UDP [10].

4. Analysis

In some works, the latency of SR-IOV is very close to that of native networking, in others it is not. In some it is not compared to native. The absolute numbers differ greatly as well, by up to 20 µs. Liu [2] measures an absolute latency of 24 µs for SR-IOV and 17 µs for native with packet sizes $\leq$1 KiB. Normalized to the latency of native networking that equates to 141 % for SR-IOV. For the same packet sizes Lockwood et al. [4] measure the latency of SR-IOV at 40 µs, which is nearly twice as much. When normalizing the latencies, SR-IOV is at about 200 % to 250 % compared to that of native. These numbers do not at all match those measured by Liu [2]. Even though the setups seem similar at first, they differ in many aspects, not even the benchmarks used to measure the latency are similar. This means that even though the numbers seem comparable, they actually are not.

Since the impact of the previously mentioned optimizations is also often not measured, we try to make some educated guesses on how it could behave The MSI and the EOI optimizations proposed by Dong et al. [3] presumably have little to no impact on latency. This is because both do not really touch when an incoming packet is processed but rather how it is processed. Since both optimizations do not massively change what actually happens when a packet arrives, we can assume that the latency stays about the same. The optimizations with the presumably largest impact on latency are the various interrupt coalescing optimizations. Since they reduce load on the CPU by coalescing interrupts together, an arriving packet might not immediately get processed. This could lead to increased latency, with the severity depending on the intricacies of the optimization. It is even touched upon by Dong et al. [3] where the degraded TCP performance after applying the AIC optimization is attributed to the higher latency sensitivity of TCP. We can assume that the other coalescing optimizations behave similarly. The CGR and AIR optimizations mentioned by Li et al. [9] and Huang et al. [10] are very similar to the previously mentioned interrupt coalescing optimizations, but they adjust the interrupt rate on the fly based on the traffic. This could mean that latency sensitive traffic is still delivered fast enough and large traffic volumes are not bottlenecked by the CPU. Though due to the way the current interrupt rate is calculated, the latency sensitive traffic would have to dominate either in number of packets or in used bandwidth. This could mean that the applications running on the machine have to be carefully chosen and matched to not interfere with each other.

5. Conclusion

As mentioned previously, the very few latency measurements which are provided paint a very inconclusive picture. The measurements which are provided can only be generalised to a certain point. We did make some educated guesses about how the optimizations mentioned before could impact latency, but those are only guesses and do not replace actual measurements. More latency measurements are definitely needed, with and without optimizations.

One question that is still left completely unanswered by any of the surveyed papers is how latency behaves with a growing number of VMs. Scalability is touched upon by Dong et al. [3] and Bauer et al. [8], but not in regards to latency. Bauer et al. [8] show what is to be expected, above a certain number of VFs the CPU becomes a bottleneck due to the huge amount of interrupts that need to be handled. Since they did not use VMs, but only VFs used by applications directly, this effect would presumably get worse since the interrupts not only need to be handled but then also virtualized. These problems could maybe be mitigated by the optimizations proposed by Dong et al. [3]. Their optimizations make VMs with SR-IOV scale nearly perfectly, at least bandwidth- and CPU-wise. Unfortunately latency is measured by neither. Either way, unfortunately, more data is also absolutely necessary.

Although we tried to focus on latency in this paper, bandwidth still needs to be talked about. There is some conflicting data regarding this. While most of the mentioned papers come to the conclusion that bandwidth does not seem to be a problem for the tested systems (mostly 10 GbE), Lockwood et al. [4] discover that SR-IOV cannot quite keep up with native. Though in this case, it could be due to the fact that MPI bandwidth is tested, not "normal" bandwidth. In most cases, SR-IOV can keep up with native, even at speeds up to 20 Gbit/s [5].

References

[1] "Xen Networking," https://wiki.xenproject.org/wiki/Xen_Networking, Last Accessed: 2021-06-07.

[2] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[3] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–10.

[4] G. K. Lockwood, M. Tatineni, and R. Wagner, "SR-IOV: Performance Benefits for Virtualized Interconnects," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, 2014.

[5] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 285–292.

[6] X. Xu and B. Davda, "SRVM: Hypervisor Support for Live Migration with Passthrough SR-IOV Network Devices," *SIGPLAN Not.*, vol. 51, no. 7, pp. 65–77, 2016.

[7] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, 2014, pp. 445–458.

[8] S. Bauer, D. Raumer, P. Emmerich, and G. Carle, "Intra-Node Resource Isolation for SFC with SR-IOV," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018, pp. 1–6.

[9] J. Li, S. Xue, W. Zhang, R. Ma, Z. Qi, and H. Guan, "When I/O Interrupt Becomes System Bottleneck: Efficiency and Scalability Enhancement for SR-IOV Network Virtualization," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1183–1196, 2019.

[10] Z. Huang, R. Ma, J. Li, Z. Chang, and H. Guan, "Adaptive and Scalable Optimizations for High Performance SR-IOV," in *2012 IEEE International Conference on Cluster Computing*, 2012, pp. 459–467.

[11] "NAPI," https://wiki.linuxfoundation.org/networking/napi, Last Accessed: 2021-07-30.

Towards General Sliding Window Stream Analysis

Simon Hanssen, Kilian Holzinger*, Henning Stubbe*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: hanssen@in.tum.de, holzinger@net.in.tum.de, stubbe@net.in.tum.de

Abstract—**Real time stream processing becomes more and more important as data arrives continuously and information needs to be based on the latest data. To query an unbounded data stream, *sliding window queries* are utilized. Naively evaluating these queries often causes a lot of redundant computations that unnecessarily lower the performance. Over the years different ideas have been proposed that capitalize on the nature of sliding window queries to reduce redundant calculations. This paper explains the basics of sliding window aggregation and then shows different techniques that emerged. These techniques are then evaluated and compared based on the performance studies conducted by the researchers and restrictions they impose on the kind of workloads they can handle.**

The techniques investigated are paned and paired Windows and a more general version of this called stream slicing. Additionally Slider and Reactive Aggregator which utilize Trees and DABA which is based on the TwoStacks algorithm are included. As of publication of this Paper, the general version of stream slicing fits best as efficient a drop-in replacement without posing any restrictions.

Index Terms—**sliding window, stream processing, stream aggregation**

1. Introduction

In many real time applications, data continuously arrives in a stream and needs to be processed as such since users want whatever information they query to be based on the most recent data. Because of this, batch processing is no longer an option. The amount of data that arrives is potentially infinite and older data might get irrelevant over time. To query an unbounded data stream *window queries* are utilized, specifying a section of the stream to be evaluated. This is usually the most recent part and as new data arrives, the query should be reevaluated. The amount of incoming data needed to trigger an update is often way smaller than all data currently relevant, so when computing a new output, many calculations are redundant, leaving opportunities for optimization. Solutions presented to capitalize on these opportunities often pose restrictions on the type of queries or streams they can be used for. This paper provides the basics needed to understand the challenges those solutions faced and then presents selected techniques, how they achieved performance gains, how they compare to previous ideas and what restrictions they pose.

The rest of the paper is structured as follows: Section 2 explains the basics of *sliding window analysis*, Section 3 defines aspects that are important when discussing the solutions in Section 4. Section 5 gives an outlook on how sliding window analysis might further evolve. In Section 6 related and relevant work not dealt with in detail in this paper is mentioned and Section 7 concludes.

2. Background

To query information from a theoretically unbounded amount of data, one uses *windows*. This means, giving a cutoff to that data is considered relevant for the query. A popular example that will be referred to repeatedly in this paper is a trader at the stock market who has access to a stream containing all trades for a specific stock. The stream is made of tuples that in this example would contain all the information about the trade they represent e.g. the amount of shares traded, the time of the trade, the price etc. They want to know the average price their stock was traded for recently and now give a cutoff for trades to still be considered. They might ask for the average price of the last 5000 trades or the average price of all trades that happened during the last ten minutes of them issuing the query. With a query like this they would define a window with a size of 5000 trades/tuples or ten minutes. The size of the window is also known as *range*

So windows are a way to query data streams, but the trader probably will issue their query more than once, as they want to have live data all the time. Maybe they ask for the average of the last ten minutes and want it updated every ten seconds. For this, *sliding window queries* exist. Here, additionally to the size of the window one is interested in, one also has to provide a measure indicating when to update the given window based on the newest state of the stream. This update measure is also called *slide*. So our trader's query would have a range of ten minutes and a slide of ten seconds.

Figure 1 shows visualizes the process of how a sliding window query behaves. The query in this example has a range of nine and a slide of three tuples respectively. One tuple is represented by one green circle, and the tupsles shown are the end of the stream. There might be an arbitrary number of tuples preceding them but they are not relevant for the query, since they are not even part of the first window. When the query first is issued the latest nine tuples are inside the window and aggregated. As time passes more tuples arrive and as soon as thrre new tuples arrived, a new aggregate is calculated (third row). The first window actually does not exit or matter anymore here, but it is left drawn to visualise the process. The forth

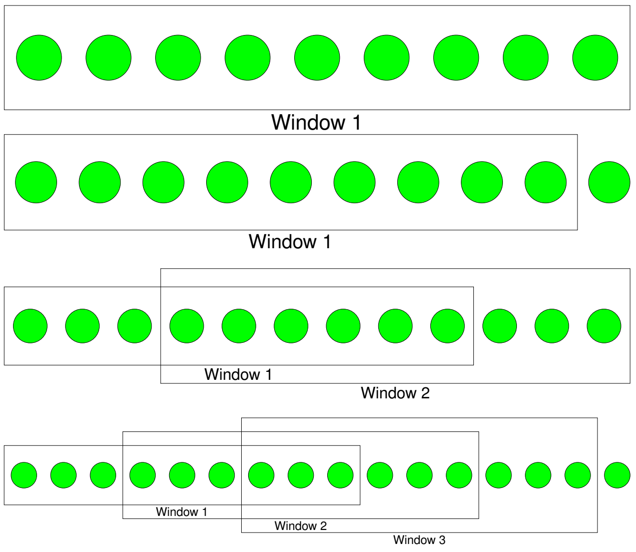


Figure 1: An example of a sliding window query with slide = 3 and range = 9 while tuples arrive over time

row shows the state after even more tuples arrived and the window is updated once more.

The naive way of answering this query would be evaluating it without any considerations about the nature of the problem: every ten seconds, find all trades in the stream that arrived at most ten minutes ago and average their price. While this might work fine for a not that frequently traded stock, it is rather inefficient in most cases. Considering that for each time the window *slides*, i.e., ten seconds pass and the new average is computed, most of the considered trades stay the same – the ones that happened in the last nine minutes and fifty seconds before the update – recomputing from scratch involves a lot of redundant calculations. Looking at Figure 1 the redundancy can be seen in the overlap of Window 1 and 2.

If we had computed the average of the last ten seconds and the other nine minutes and fifty seconds separately and then combined them, we could use that again with the new ten seconds of data that arrive. The next time ten seconds pass, we would have to recompute everything again though, since we cannot easily split off another ten second slice of the nine minute and fifty second chunk left over. This nevertheless shows the potential for optimization, later in the paper we will see how different techniques capitalize on that.

3. Workload Characteristics

Before investigating the different solutions that were developed, requirements different workloads pose to stream processing systems will be given here to allow the discussion of the solutions later.

3.1. Latency and Throughput

Naturally, memory usage and computation time are two important factors when it comes to a good query engine, but the time aspect needs to be viewed from two points here: latency and throughput. The former describes the time it takes a query to be answered once it is issued, the latter the amount of data that can be handled in a certain amount of time, which is directly dependent on how long it takes to process an incoming tuple.

3.2. Requirements to the Aggregation Functions

In general a sliding window query provides windows of tuples repeatedly and then performs some sort of operation on those tuples to gather an useful output from them. These operations can be seen as aggregation functions taking a set of tuples as input and giving the desired result as output. In the case of our stock trader that function would take the average price of all trades in the set given to it. When discussing aggregation functions in the context of stream processing, it is important to note that they are not seen as operations on sets of tuples. Instead they are split into three sub-functions. These sub-functions will be explained using *average()* as an example: (1) The first function takes one data tuple and converts it into a partial aggregate. In the case of *average()* this would take the input and convert it into a tuple *(sum,count)* with the value of the input as *sum* and 1 as *count*. (2) The second function takes two of those partial aggregates and combines them into one. In our case that would be adding up the *sum* and *count* parts of the partials respectively. (3) The last function takes one partial aggregate and converts it into a final output, here it would return *sum/count*, this would then yield the average of all tuples that were combined into the partial aggregate used as input for the third function.

When discussing aggregate functions, mainly the second function is of interest, so "*average()* is commutative" means that the combination of two partial aggregates is. This idea is crucial for many techniques presented later, since it allows partial (pre-)aggregation. This only works as long as the function is associative though, which is why all ideas presented later require this to be the case. Considering the lack of literature for cases where it is not given and authors noting that cases where it is not given are rare, associativity seems to be a reasonable assumption to make [1]–[3].

Invertibility and commutativity on the other hand are not given for every function of common interest; functions like *max* and *min* are not invertible. As an example for a not commutative aggregate Tangwongsan et al. name "collect-like" functions like *concatStrings* [2].

3.3. Requirements to the Stream

An important requirement several solutions have is that the tuples in the stream arrive in order. To show why this can be crucial, we can look at the stock trading example again. If the data about the trades does not arrive in the order that the trades took place and we also do not sort them on arrival, answering some queries can become difficult. If our stream is ordered, finding an entry that is older than ten minutes signals that up until that point, all entries about trades are within our range of interest. If we cannot guarantee that the stream is in order though, the entry might just have arrived a little late for some reason, there might be further entries of interest further behind. Maybe the internet connection in a trading hub went down, and now information about a trade that happened arrives fifteen minutes late. Under the assumption that the stream is in order, one would now stop looking further for relevant trades and get a skewed result. Assuming the stream is not in order one would struggle finding a point where it is guaranteed that no relevant trades can be found in the rest of the stream anymore.

3.4. Requirements to the Window

Different solutions have varying restrictions on the windows they can handle. A good way to classify those is presented by Traub et al. [4]. They divide windows into three categories:
(1) **Context free windows**: a window is context free, if its boundaries are already defined before it begins. The first stock trading example (average of the last ten minutes every ten seconds) is context free, all windows start ten seconds apart and span ten minutes, independent of the trades that happen.
(2) **Forward context free windows**: a window is forward context free, if we can tell whether a tuple marks the beginning or end of a window as soon as it arrives. An example: our stock trader wants to know the amount of trades that happened, separated each time the price of the stock passes the $100 mark. We cannot tell when the window will end until a trade happens that crosses the mark, but for every trade we already know of, it is clear whether or not it closed the preceding window.
(3) **Forward context aware windows**: for a forward context aware window, we cannot tell if a tuple marks the beginning of a window in the moment that it is processed. The second version of the stock trading example (average of the last 5000 trades every ten seconds) falls into this category. Until ten seconds have passed and the window slides, it is unclear which tuples are within the cutoff of the 5000 tuple range we specified. This is because each arriving tuple pushes the the cutoff further, and only after the ten seconds are over we can be sure that no tuples relevant for that update will arrive.

Another occurring aspect are concurrent windows. The same or different users might issue more than one query over the same data stream with different slide and range parameters, forcing parallel evaluation of each of those on their own if not taken into consideration. While this does not impose any restrictions if the system is aware of concurrent queries existing it can capitalize on that for further improvements.

4. Techniques

In this section different techniques for speeding up stream processing in comparison to from scratch recomputation will be presented.

4.1. Paned Windows

A first step to prevent having to recompute the whole aggregate every time the window slides was made with the introduction of *paned windows* by Jin Li et al. [5] They split the arriving data stream up into smaller sections - *panes* - of the same size, choosing the size as the greatest common divisor of the *slide* and *range* of the window. Using the first stock trading example this would mean creating panes with the size of ten seconds. For each of these panes the partial aggregate is computed, and when the next full aggregate is needed, only these partial aggregates need to be combined. This is beneficial in two ways: the partial aggregate for one pane only needs to be computed once and can be used again (remember: nine minutes and fifty seconds of the window stay the same each time it slides) and when the final aggregation is due, most of the work has already been done by computing the partial aggregates. For calculating and combining the partial aggregates the associativity of the function is crucial.

Once a pane has been aggregated, the tuples making it up are not required to stay in memory anymore, allowing for savings here, too. Taking our example again, instead of having to save the thousands of trades that happened in the last ten minutes, only 60 partial aggregates are needed.

This is the first instance of a technique that will later become known as *stream slicing* [4], [6]. Several other ideas presented later pick up on the idea and improve it, overcoming the restrictions that paned windows still have: (1) windows need to be *context free* so the size of the panes can be determined. (2) The stream needs to be in order as arriving tuples are inserted into the currently active pane.

While the performance evaluation conducted was not very thorough, in the cases that are relevant for actual applications, i.e., more than a few tuples per pane and panes per window, they find a speedup of 5 to 10 times when processing the query compared to recalculation. The greater the number of tuples per pane and panes per window, the better and as Krishnamurthy et al. noted it is reasonable to expect those numbers to be large enough for significant efficiency gains in real life scenarios [7].

4.2. Sharing Paired Windows and Fragments

Krishnamurthy et al. pick up the idea of paned windows and improve and extend it in the following ways: (1) They introduce *paired windows*, a way to slice a stream into fewer slices than when using paned windows, allowing for faster final aggregation. (2) They present a way of handling multiple sliding window queries over the same stream efficiently. They do this by slicing the stream so that the partial aggregates can be *shared*, i. e., used by all queries. While this leads to smaller and therefore more slices, it prevents redundant computations because in a non-sharing case every query would calculate the aggregates leading to the slices independently. (3) They introduce *shared Data Fragments*, a way to allow different selection predicates in concurrent queries while still taking advantage of *sharing* partial aggregates [7].

So if one trader only wants to consider trades where more than 100 shares were traded, and another one is only interested in those where the price was at least $4000 in total, their queries do not need to be handled separately anymore. While this idea is not discussed in further literature, it should be easy to include it in other stream slicing techniques like the one described in Section 4.6. This is because the splitting in shards happens after and independently of the slicing, so no matter what slicing technique is used one can split up the resulting slices afterwards.

While their analysis shows that paired windows only offer a minor improvement to paned ones, when sharing partial aggregates between concurrent windows with a regular workload, their implementation only needed 10% of the time to calculate aggregates.

4.3. Slider

Bhatotia et al. present an idea based on the idea of *incremental computing* [1]. They introduce *self-adjusting contraction trees*, a set of data structures that is able to deal with the requirements of inserting and removing tuples constantly as sliding windows demand. The leaf nodes of these trees are the output of applying the first aggregation sub-function to the tuples arriving from the stream and each inner node contains the partial aggregate resulting from combining its children. The value in the root node is then used to compute the final aggregate. They implemented a system they named *Slider* that uses these trees to efficiently evaluate sliding window queries.

The resulting speedup is only mediocre though; compared against recomputation from scratch their tests showed a speedup of up to four times. While they did not include a reference implementation of the solutions presented earlier in their tests and they use other metrics to evaluate, results presented in other papers indicate a better performance of those compared to Slider [1], [4], [5], [7], [8]. Furthermore Slider does not allow concurrent windows and one of their optimizations requires the aggregation function to be commutative, which is, as explained, not always the case. Slider accepts queries stated in *Pig-Latin* [9] allowing all types of windows to be utilized [1]. Like with the previous ideas, the stream needs to be ordered here, too.

4.4. Reactive Aggregator

Tangwongsan et al. present a very similar approach to Slider [3]. They create a binary tree on top of all tuples currently in the window. For this they created the *FlatFAT* data structure which stands for *flat fixed-size aggregator* and the *Reactive Aggregator (RA)* framework which uses FlatFAT to efficiently evaluate window queries. They show that their implementation needs at most $\mathcal{O}(m + m \log(n/m))$ partial aggregations for an update of size *m* to a window of size *n*. They also only compare their implementation against one that recomputes everything from scratch every time and use a *slide* of 1 for those comparisons, showing that, in that case, their solution becomes more than 10x faster than recomputation. They reason that this slide granularity is the worst for their implementation but the same goes for recomputation. Because of the similarity of the ideas, the real performance gains probably are comparable to Slider.

Traub et al. included an implementation of RA into performance studies they made in [4], [6]. It showed excellent latency but because each new arriving tuple forced the binary tree to be updated, the throughput suffered compared to slicing techniques. In comparison to Slider they do not require commutativity of aggregate functions in any way. They also allow tuples to be *evicted* out of order. While this can be useful in some cases, the way more important case of *inserting* tuples out of order was not addressed. Since RA only allows aggregation on all tuples currently handled, it does not support concurrent queries, but at the same time allows all types of windows to be used.

4.5. DABA

The *De-Amortized Banker's Aggregator* or *DABA* is an algorithm developed by Tangwongsan et al. [8] It guarantees worst case constant time for each window operation, without requiring the aggregation function to be invertible and by this allowing for consistently low latency. For an invertible aggregation function this is easy to achieve: adding a tuple just means aggregating it with what already has been accumulated and removing one uses the inverted function with the tuple to be removed. DABA uses a system of pointers and partial aggregates to allow this for non-invertible functions as well. DABA is based on an algorithm called *Two-Stacks* that only had amortized constant cost, causing occasional peaks in latency. The modifications made allow the work to be spread across all operations made.

In their performance study these effects show: while Two-Stacks has a lower average latency, the standard deviation of latencies of single operations was about 20 times higher compared to DABA. The overhead for spreading out the work between all operations results in slightly lower throughput for DABA. It still outperforms Reactive Aggregator regarding throughput while the only restriction compared to RA is that windows must be FIFO, but since it is unusual for tuples to be evicted out of order, this is usually the case.

4.6. General Stream Slicing

Traub et al. developed a generalization of stream slicing techniques that removes all restrictions given except the associativity of the aggregation function [4]. The basic idea stays the same: the stream is split up into smaller slices that do not need to be divided further because there are no window borders within them. Because of that they can be partially aggregated allowing working on slice basis instead of tuple basis.

They present a set of decision trees that based on properties of the windows, stream and aggregation function allow to decide, the best way to handle a query for a specific case. One example: for an in-order stream with forward context free windows, it is sufficient to only keep the partial aggregates of slices in memory, allowing tuples to be discarded after they were handled and by this reducing memory usage.

They implemented an eager and a lazy version; the eager one computes a tree based on Reactive Aggregator, but with the slices as leaves instead of tuples. By doing this it provides great latency while not suffering from as massive drawbacks in throughput as RA compared to the lazy version since the tree is way smaller.

The results from their experiments show great potential, general stream slicing matches or outperforms other techniques they compared it to even if the conditions for those were optimal.

5. The Future

While work in the past mainly was focused on finding new ideas that allow for more efficient stream processing than naively recomputing everything when needed and lowering restrictions those ideas posed, we now arrived at a state where a general approach has been found. It already incorporated different concepts from before but there is still room for improvement. As Tangwongsan et al. lately suggested, DABA could be used in general stream slicing if the stream is in-order [10]. So instead of improving each technique on its own or coming up with new ones, combining the different strengths of different approaches seems promising.

6. Related Work

Cutty [6] was another step towards general stream slicing that picked up on some aspects introduced by RA. Scotty [11] is an open-source implementation of general stream slicing. FiBA [2] uses finger trees to allow efficient handling of out-of-order tuples, showing excellent results on its own and being another attractive candidate to include in general stream slicing to improve the out-of-order case [11]. Zhang et al. analyzed different techniques that focus on utilizing hardware as well as possible for fast stream processing [12].

7. Conclusion

Optimizing sliding window aggregation poses different challenges that can prevent optimizations to reduce often occurring redundant computations from being used in general stream processing systems. With the introduction of paned windows came the idea of *stream slicing*, i.e., the realization that one can take partial aggregates of several tuples as a new smallest unit and still compute everything needed. This came with a rather strict set of restrictions though. Later this concept was generalized lifting those restrictions. This general version already performs well and on top of that offers opportunities to include other specialized optimizations like DABA, RA or FiBA to further increase the speed of the different cases that need to be handled. After a lot of ideas that rather were a proof of concept than really applicable, with this there now exists an efficient alternative to conventional, recomputing-based solutions that can be used as a drop-in replacement with potential to increase performance even further with more research done.

References

[1] P. Bhatotia, M. Dischinger, R. Rodrigues, and U. A. Acar, "Slider: Incremental sliding-window computations for large-scale data analysis," *MPI-SWS, CITI/Universidade Nova de Lisboa, CMUTechnical Report: MPI-SWS-2012-004 September*, 2012.

[2] K. Tangwongsan, M. Hirzel, and S. Schneider, "Optimal and general out-of-order sliding-window aggregation," *Proc. VLDB Endow.*, vol. 12, no. 10, p. 1167–1180, Jun. 2019. [Online]. Available: https://doi.org/10.14778/3339490.3339499

[3] K. Tangwongsan, M. Hirzel, S. Schneider, and K.-L. Wu, "General incremental sliding-window aggregation," *Proceedings of the VLDB Endowment*, vol. 8, no. 7, pp. 702–713, 2015.

[4] J. Traub, P. M. Grulich, A. R. Cuéllar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl, "Efficient window aggregation with general stream slicing." in *EDBT*, 2019, pp. 97–108.

[5] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "No pane, no gain: efficient evaluation of sliding-window aggregates over data streams," *Acm Sigmod Record*, vol. 34, no. 1, pp. 39–44, 2005.

[6] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, and V. Markl, "Cutty: Aggregate sharing for user-defined windows," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1201–1210.

[7] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006, pp. 623–634.

[8] K. Tangwongsan, M. Hirzel, and S. Schneider, "Low-latency sliding-window aggregation in worst-case constant time," in *Proceedings of the 11th ACM international conference on distributed and event-based systems*, 2017, pp. 66–77.

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: A not-so-foreign language for data processing," ser. SIGMOD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1099–1110. [Online]. Available: https://doi.org/10.1145/1376616.1376726

[10] K. Tangwongsan, M. Hirzel, and S. Schneider, "In-order sliding-window aggregation in worst-case constant time," *CoRR*, vol. abs/2009.13768, 2020. [Online]. Available: https://arxiv.org/abs/2009.13768

[11] J. Traub, P. M. Grulich, A. R. Cuéllar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl, "Scotty: General and efficient open-source window aggregation for stream processing systems," *ACM Trans. Database Syst.*, vol. 46, no. 1, Mar. 2021. [Online]. Available: https://doi.org/10.1145/3433675

[12] S. Zhang, F. Zhang, Y. Wu, B. He, and P. Johns, "Hardware-conscious stream processing: A survey," *SIGMOD Rec.*, vol. 48, no. 4, p. 18–29, Feb. 2020. [Online]. Available: https://doi.org/10.1145/3385658.3385662

Tracing the Execution Path in Mac80211

Pooja Parasuraman, Jonas Andre*, Stephan Günther*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: pooja.parasuraman@tum.de, andre@net.in.tum.de, guenther@tum.de

Abstract—**The purpose of this study is to trace the execution paths of a wireless packet, traversing within the Mac80211 subsystem. Eventhough the world progresses towards building high speed wireless networks by optimizing link and routing costs, the end system's processing capability remains a bottleneck in limiting the efficiency of networks. As first step in optimising per-packet processing at wireless endpoints, we analyse the existing architecture of the Mac80211 subsystem. We discuss the transmission path of IEEE802.11 packets with a focus on managed mode of operation. This paper presents the structure and design of the Linux Kernel and the functions through which IEEE802.11 packets traverse. An experiment is perfomed to extract real-time trace of packets using explicit kernel logs for comparing the results obtained from manual tracing of Linux source code.**

Index Terms—**Linux Kernel, Mac80211, Wireless Driver, IEEE802.11 WLAN**

1. Introduction

One of the most widely used protocols in the family of IEEE 802 Local Area Network standards is the wireless LAN protocol (`IEEE 802.11` [1]). `IEEE 802.11` standard defines the link layer and physical layer protocols for communication between wireless devices. This standard needs to be followed by all driver developers in order to facilitate interoperability.

The Linux operating system provides a generic framework for wireless device drivers called the Mac80211. It is a subsystem that interfaces between the kernel and the device driver for various functionalities with respect to the wireless network packets that pass through the subsystem. An `IEEE 802.11` wireless network interface card (NIC) can operate in one or many modes [2] as discussed below.
Master : NICs in Master mode act as Access Points (AP) and follow a hierarchy of operation. A connection to another wireless NIC is possible only if the latter operates in Managed mode. This mode can also be termed as AP mode or Infrastructure mode.
Managed : Managed mode NICs act as clients (also termed as slaves) and associates to an already created wireless network by a master card. Managed mode is the counter-part of Master mode. There is a strict master-slave hierarchy and hence a client NIC can only communicate with its own master. It is not possible for two client cards to interact between themselves directly.

There are other less commonly used modes in which a NIC can be configured. In Monitor mode, NICs can sniff all radio traffic on a particular channel for wireless network debugging and analysis. Promiscuous mode is similar to Monitor mode but the difference is the former mandates an association with an AP (active sniffing) while the latter supports passive sniffing. Ad-hoc mode is used in peer-to-peer networks. Mesh mode combines ad-hoc mode and routing. A NIC in Repeater mode extends the existing wireless networks for longer range of access. In Tunnelled Direct Link Setup (TDLS) mode, a direct secure fast path for data transfer between communicating peers is made possible in a hierarchical network. This facilitates faster media streaming and other data transfers.

This study is based on manual tracing using the open-source Linux Kernel source code with focus on Managed mode. Section 2 talks about the related research work performed in this area. Section 3 provides an overview of the Linux Kernel architecture with respect to the Wireless network stack. Section 4 talks about Mac80211 subsystem in detail. A special mode available in Mac80211 subsytem of Linux Kernel - the Fast Xmit mode - is discussed in Section 5. Finally, Section 6 explains the experiment conducted to trace `IEEE802.11` packets traversing the Mac80211 subsytem.

2. Related work

Vipin et al. analyses the implementation of `IEEE802.11` network stack in the Linux Kernel and its interaction with open source device drivers [3]. Lisovey et al. discuss about the feasibility of including a module in the Linux Kernel to enable wireless communication for vehicular environment. Vitalik et al. proposes a design for portable and pluggable mode for the Mac80211 subsystem with add-on features such as co-operative retransmission support [4]. All these papers, analyse the wireless network stack in the Linux Kernel and discuss whether the design of Mac80211 framework can be optimised and enhanced.

The work from multiple analysis, implementations and architectural papers have been used as a base for this paper in order to understand the architecture of the Mac80211 subsystem.

3. Linux Kernel and Mac80211

The Linux Kernel network stack is designed in a modular way with a clear separation between multiple entities. Figure 1 depicts how the wireless network stack is layered in the Linux Kernel and how interaction between the layers takes place. The core understanding of this architecture is obtained from [5] and [6].

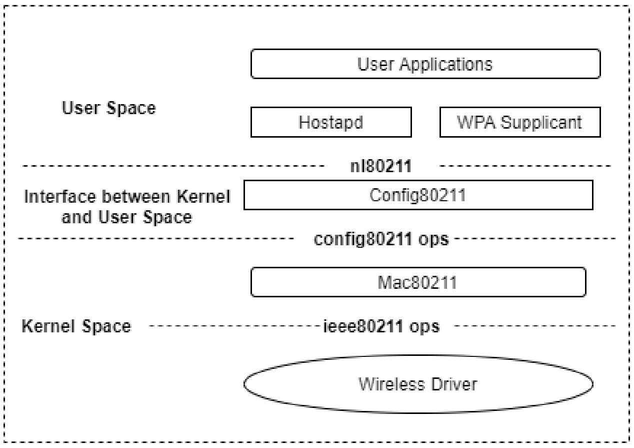


Figure 1: Kernel Wireless Stack

In order to better understand the architecture, we analyse each component using a top-down approach. The user space is composed of applications, such as `Hostapd` and `WPA supplicant`, that use wireless as their underlying technology. `Hostapd` facilitates configuration and control of Access Points, whereas `WPA supplicant` is a control interface to manage wireless clients. User space applications interact with the Linux Kernel using config80211 subsystem [7]. Config80211 is a configuration API acting as a bridge between user space applications and the underlying driver via the Mac80211 subsystem. Interaction between user space and the config80211 happens via the 802.11 netlink interface called nl80211 [8].

The Mac80211 subsystem [9] is a generic framework provided by the Linux Kernel. Wireless drivers use the Mac80211 framework to register callbacks for all packet processing functionalities. These are available in kernel space and they interact using config80211 ops and ieee80211 ops [8], which are the wireless configuration operations. Every wireless interface has its own set of configuration operations which is called from user space.

4. Mac80211 TX Path

This section provides a detailed schema of the Mac80211 subsystem [9] with respect to managed mode. Understanding of the information in this section is obtained by manual tracing backed by the experiment discussed in Section 6.

Figure 2 shows the path (functions) taken by a wireless packet from user space to the driver. These functions correspond to handling packets when the sending interface operates in managed mode.

A packet is originated by a user space application and sent to the kernel. The packet then passes through various OSI layers [10] of the kernel stack to add layer specific information. Packets are carried within a data structure called skb (socket buffer) [11] and are passed onto every layer in the kernel.

All the relevant information required for the higher layer OSI headers [10] is added to the skb by the Linux Kernel. The kernel then hands the skb to the Mac80211 subsystem along with the information of the interface via which the packet must be sent (outbound). This is achieved by a registered callback function (ndo_start_xmit()).

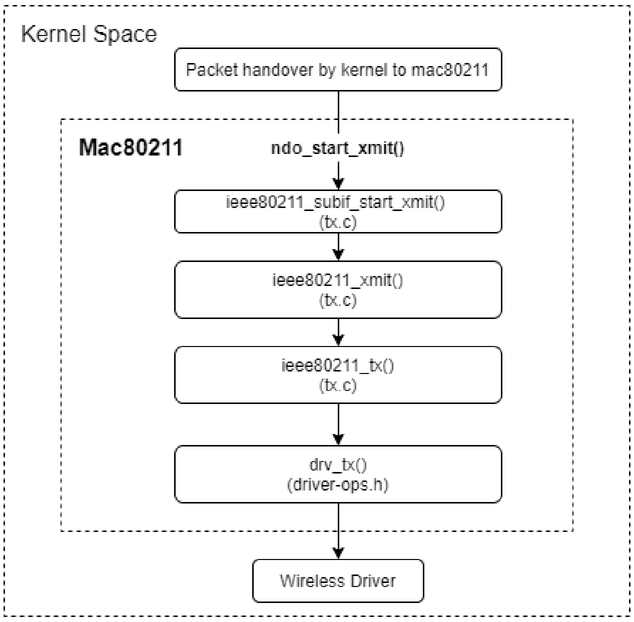


Figure 2: Mac80211 Control Flow

The corresponding registered function for ndo_start_xmit() with respect to managed mode is ieee80211_subif_start_xmit(). This function is responsible for the actual MAC layer processing and adding MAC and PHY information into the skb. After the required information is filled in, it is passed onto the ieee80211_xmit() function which handles the adjustment of skb headroom and sets the QoS header, if required. The skb is then passed onto the ieee80211_tx() function. Every interface has its own transmission (TX) queue. In order to transmit the skb into the correct interface, corresponding transmission queue of the outbound interface is selected. The skb is put into the outbound interface's TX queue. The skb will be dequeued and sent to the final point in the Mac80211 subsystem (drv_tx()) by a kernel tasklet named ieee80211_tx_pending. At the end of drv_tx() the control is transferred from the Mac80211 subsystem to the wireless driver.

ieee802111_subif_start_xmit

The majority of the MAC layer processing happens within the ieee80211_subif_start_xmit() function. Figure 3 shows the detailed flow diagram of control points in this function. The Mac80211 subsystem maintains a hash table with a list of known stations connected to the local device. The initial step is to retrieve the target station (STA) node from the hash using the target address from the skb data structure. In managed mode, various parameters must be verified to retrieve the target station node. It is checked if the target station is a TDLS peer to the local device. If it is a TDLS peer, then more checks are done to verify if the target station is an authenticated peer to send or receive data to/from the local device. If all the above conditions are satisfied, the station node is retrieved with target mac address as the key to the hash table. If the STA node is found successfully, the skb is attempted to be sent via a

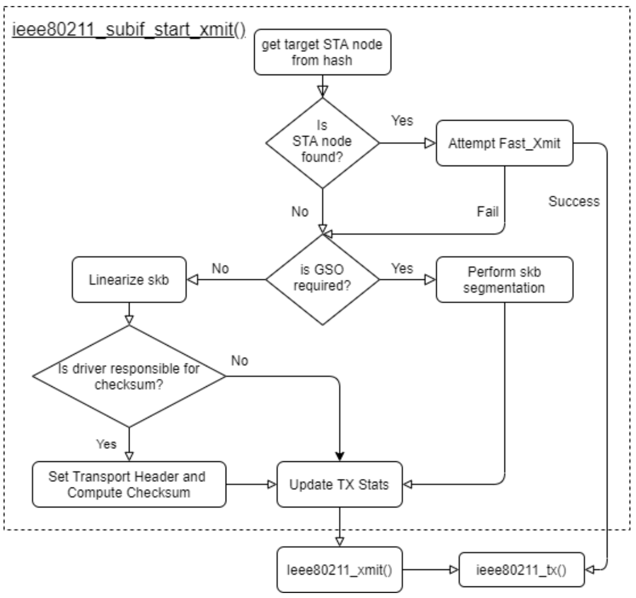


Figure 3: Flow Diagram of ieee80211_subif_start_xmit function

relatively faster execution path, called the fast_xmit path [?]. This is explained in detail in Section 5.

If the station node is not found in the hash table or the skb could not be sent via the fast_xmit path, then the skb takes regular slow path where detailed checks are made before sending the packet out through the interface. In this path, checks to verify if the skb must undergo Generic Segmentation Offloading (GSO) [12] is performed. GSO is a software-based technique to perform packet segmentation which is offloaded by wireless cards to drivers. If a skb is subject to GSO, then it will be segmented into multiple skbs based on the MSS segment size provided by the wireless card using gso_size config parameter. The segments will then be transmitted onto the interface.

If a skb does not require GSO, the skb must undergo linearization which is a process in which, a paged skb is converted to a linear skb [11]. A paged skb is used when the data that needs to be sent is larger than the MSS. One predominant use case of a paged skb is when a file system file content needs to be sent over a socket. Once, the skb is linearized, it is checked if the wireless card has offloaded the checksum [13] operation to the driver. The flag CHECKSUM_PARTIAL specifies if checksum needs to be verified in the software. If the flag is set, then the transport header offset is adjusted such that there is enough space for the checksum to be inserted. The checksum is calculated and copied into the skb after the header size is modified. With this, all the necessary processing is done and the TX statistics are updated for TX packet count and TX byte count for the interface. The skb is then sent to the ieee80211_xmit() function which is explained above.

5. Fast Transmit Path

This section discusses one of the interesting features of the Mac80211 subsystem - Fast Transmit Path [14]. The understanding of information provided in this section is based on manual code analysis with the open-source Linux

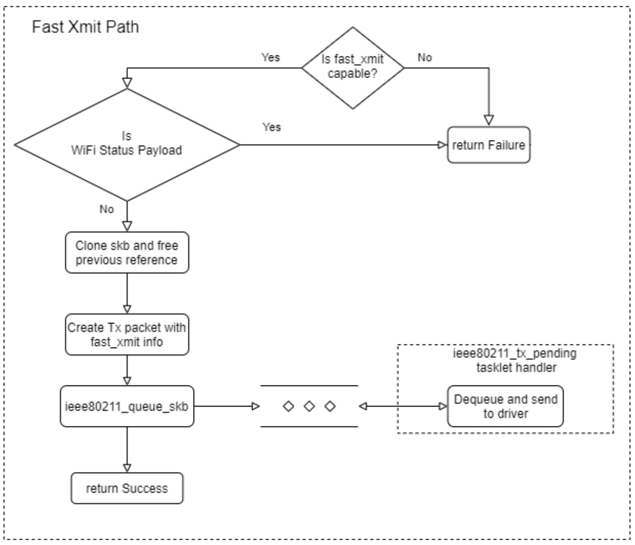

Fast Xmit Path
Is fast_xmit capable?
Yes
No
Is WiFi Status Payload
Yes
return Failure
No
Clone skb and free previous reference
Create Tx packet with fast_xmit info
ieee80211_queue_skb
ieee80211_tx_pending tasklet handler
Dequeue and send to driver
return Success

Figure 4: Flow Diagram of Fast Transmit Path

distribution [8]. The Fast Transmit feature requires support from hardware. Figure 4 shows the list of functions a packet takes in Fast Transmit mode.

Fast Transmit is used as an optimization technique so that packets do not need to go through a long list of checks as discussed in Section 4. Before a packet is sent for further processing, the function ieee80211_check_fast_xmit() checks if the target station and the local device's underlying wireless card provide Fast Transmit support in their hardware. If Fast Transmit is possible, further checks to determine if the target station is an authorized peer are done to proceed with Fast Transmit. Once these checks are passed successfully, the 802.11 header and other information for packet processing is cached inside the STA node data structure, which is required during TX packet processing. This function is called whenever a new station is added or any state change happens at the station. Explicit calls to this function must be made to reset the information in case the fast_xmit path is no longer applicable for the station.

During TX packet processing, the STA node retrived from hash table is verified for the fast_xmit information. If the information exists (not NULL), then the skb is capable to be sent via the fast_xmit path, else the skb follows the regular path as discussed in Section 4. Once the skb enters the fast_xmit path, the protocol of the packet is checked. If the skb is a Wi-Fi Status message, then the skb is sent to be processed via the regular path, else it is further processed as a Fast Transmit packet.

The final packet is constructed using the previously cached 802.11 header information. Once the packet is ready to be sent out of the interface, a TX queue slot for the outbound interface is fetched and the packet is put into the queue. As discussed earlier, the ieee80211_tx_pending tasklet takes care of dequeuing the skb and passing it onto the driver.

6. Packet Tracing with Kernel Logs

We performed an experiment to trace IEEE802.11 packets passing through the Mac80211 subsystem in the

Linux Kernel. Figure 5 shows the setup used for the experiment. The setup includes Raspberry Pi 4 and an external RT5572 Wireless adapter. The Raspberry Pi 4 is configured as an ethernet backhaul extender. A backhaul is an interface through which a device can connect to another existing network. This means that the Raspberry Pi 4 is now acting as a bridged wireless access point within our already existing local Ethernet network. The external RT5572 Wireless adapter is attached to the Raspberry Pi 4 and is considered as the Access Point in our experiment. The backhaul interface of the Raspberry Pi 4 is connected to a D-Link Router. We use an Android mobile phone as our client device which is connected to the RT5572 wireless adapter.

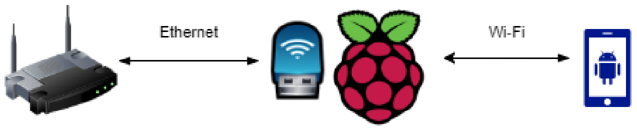
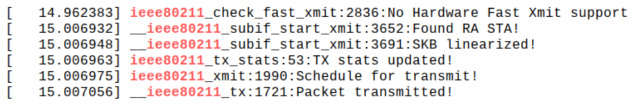


Figure 5: Experimental Setup

Explicit kernel logs were added to the open source Mac80211 subsytem source code to help us trace the packets within the Linux Kernel. We sent ICMP echo messages [15] from the Raspberry Pi to the Android phone. Figure 6 shows a sample trace captured during the experiment. The trace shows list of functions traversed by a single IEEE802.11 TX packet from the Raspberry Pi 4.



```
[   14.962383] ieee80211_check_fast_xmit:2836:No Hardware Fast Xmit support
[   15.006932] __ieee80211_subif_start_xmit:3652:Found RA STA!
[   15.006948] __ieee80211_subif_start_xmit:3691:SKB linearized!
[   15.006963] ieee80211_tx_stats:53:TX stats updated!
[   15.006975] ieee80211_xmit:1990:Schedule for transmit!
[   15.007056] __ieee80211_tx:1721:Packet transmitted!
```

Figure 6: Packet Trace

As first step discussed in Section 4, the kernel hands over the TX packet to mac80211 subsytem. Then check to find if the target station is available in the cached memory (hash table) is done. The kernel log "Found RA STA" denotes that the station is already known to the local device and its information is available in the hash table. Now, the Mac80211 subsystem checks if the packet can take Fast Transmit path. Both the RT5572 adapter and the client device did not exhibit support for Fast Transmit. Hence the kernel log "No Hardware Fast Xmit support" appears in our trace. Hence, The Tx packet follows the regular slow path.

The next step is to check if the hardware has offloaded segmentation to the driver. The RT5572 adapter takes care of segmentation in the hardware and hence GSO path is not triggered. Alternately, the skb undergoes linearization. We introduced kernel logs to find if the Mac80211 subsystem is responsible for handling checksum. Since those logs did not appear in our packet trace, it is understood that the hardware takes care of calculating checksum before sending the packet into the network. In the final step of processing, the TX statistics are updated for the TX packet which is seen from the log "TX stats updated". From this point, the underlying IEEE802.11 device driver takes care of sending the packet to the target station.

The prepared skb is enqueued in the TX queue of the outbound interface by the device driver. After this, the tasket handler discussed in Section 4 dequeues the skb and transmits the packet onto the interface which is seen in the packet trace. With this the entire lifetime of a packet within the Mac80211 subsystem is traced.

7. Conclusion and Future Work

This paper provides a detailed walkthrough of the Mac80211 subsystem and its architecture focussing on packet processing in managed mode of operation. This papers also analyses the fast_xmit mode of transmission which helps in drastically reducing the per-packet processing overhead. An experiment to back the information obtained from manual tracing is performed and the complete packet trace is presented. Future work can include finding possible areas of optimizations and analysing the remaining features of the mac80211 subsystem in order to create a simpler and a pluggable version of the mac80211 subsystem. Furthermore, the IEEE802.11 driver can be examined and a detailed study can be made on how packets traverse through wireless drivers.

References

[1] G. Hiertz, T. Denteneer, L. Stibor, Y. Zang, X. Costa-Pérez, and B. Walke, "The ieee 802.11 universe," *Communications Magazine, IEEE*, vol. 48, pp. 62 – 70, 02 2010.

[2] "Monitor mode," https://en.wikipedia.org/wiki/Monitor_mode, [Online: accessed 06-June-2021].

[3] M. Vipin and S. Srikanth, "Analysis of open source drivers for ieee 802.11 wlans," in *2010 International Conference on Wireless Communication and Sensor Computing (ICWCSC)*, 2010, pp. 1–5.

[4] V. Nikolyenko and L. Libman, "Coop80211: Implementation and evaluation of a softmac-based linux kernel module for cooperative retransmission," in *2011 IEEE Wireless Communications and Networking Conference*, 2011, pp. 239–244.

[5] D. C. Mur, "Linux wi-fi open source drivers," http://www.campsmur.cat/files/mac80211_intro.pdf, [Online: accessed 06-June-2021].

[6] J. M. Berg, "Mac80211 overview," https://wireless.wiki.kernel.org/_media/en/developers/documentation/mac80211.pdf, 2009, [Online: accessed 06-June-2021].

[7] "Linux 802.11 driver developer's guide," https://www.kernel.org/doc/html/v4.12/driver-api/80211/index.html, [Online: accessed 06-June-2021].

[8] "Linux kernel developer documentation," https://wireless.wiki.kernel.org/en/developers/documentation, [Online: accessed 06-June-2021].

[9] P. Salvador, S. Paris, C. Pisa, P. Patras, Y. Grunenberger, X. Perez-Costa, and J. Gozdecki, "A modular, flexible and virtualizable framework for ieee 802.11," in *2012 Future Network Mobile Summit (FutureNetw)*, 2012, pp. 1–8.

[10] J. Day and H. Zimmermann, "The osi reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[11] "How skbs work," http://vger.kernel.org/~davem/skb_data.html, [Online: accessed 06-June-2021].

[12] "Segmentation offloads in the linux networking stack," https://www.kernel.org/doc/Documentation/networking/segmentation-offloads.txt, [Online: accessed 06-June-2021].

[13] T. C. Maxino and P. J. Koopman, "The effectiveness of checksums for embedded control networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 1, pp. 59–72, 2009.

[14] "Fast transmit," https://elixir.bootlin.com/linux/latest/source/net/mac80211/tx.c#L2908, [Online: accessed 06-June-2021].

[15] "Internet Control Message Protocol," RFC 792, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc792.txt

TCP Congestion Control Fingerprinting

Kevin Ploch, Benedikt Jaeger[*]
[*]*Chair of Network Architectures and Services, Department of Informatics*
Technical University of Munich, Germany
Email: kevin.ploch@tum.de, jaeger@net.in.tum.de

***Abstract*—In order to make judgements on the spread of certain congestion control algorithms a way to fingerprint the algorithm of a host is needed. This can be done with congestion control identification (CCI) algorithms. This work presents the general approach of such an algorithm and summarizes possible categorizations for CCI. It presents Congestion Avoidance Algorithm Identification (CAAI) as an active and DeePCCI as a passive example. In an accuracy evaluation over a WAN connection these algorithms are compared to each other and to a more recent approach from 2020, Inspector Gadget (IG), which includes further optimizations. IG shows near perfect accuracy, DeePCCIs and CAAIs accuracies are rather humble, former with 90-92% and latter with 41-94%, and to conclude we explore how these results come to be.**

***Index Terms*—congestion control, congestion control identification**

1. Introduction

The biggest share of Internet traffic is under control of the Transmission Control Protocol (TCP) which combines concepts to provide properties like reliable and ordered data delivery or management of the sending-rate. The most performance significant part in data transmissions is the latter, namely Congestion Control (CC). Historically many TCP CC algorithms have been developed trying to find the best balance between bandwidth utilization and congestion in the network. While TCP Reno or TCP Cubic have been a wide-spread standard, whether now or in the past, newer algorithms like TCP BBR make their way into modern operating systems. [1, Section 1] [2]

To be able to make judgements on the current state of the Internet in topics like TCP performance, especially inter-algorithm-performance, or stability it is important to know the widespread adoption of each specific CC algorithm. Further examples that depend on this knowledge are topics like buffer sizing of routers, active queue management, fairness tuning of new CC variants or the creation of realistic traffic generators. To gather this needed, wide-spread deployment data we need a way to fingerprint the CC algorithm of a single host, leading us into the realm of Congestion Control identification (CCI) algorithms. [2], [3]

This paper aims to give an overview on current TCP CCI approaches. Section 2 revises important CC terminologies and differentiates the most important CC algorithms. Section 3 explains the general approach of a CCI method and presents different categories for them. Section 3.1 and Section 3.2 explain two unique approaches to CCI. Section 3.3 presents a more recent work with optimizations based on the two presented previous methods. Section 4 evaluates the accuracy of the three approaches and aims to explain the differences and pitfalls.

2. Background

Transport protocols determine the sending rate and have to balance between full utilization of network ressources and fairness among connections sharing a bottleneck link. Uncontrolled flows that exceed the speed at which a router can process packets leads to the build-up of packet queues and finally to dropped packets as the routers memory is exceeded [1, Section 2]. To accomplish this balance the protocol has to dynamically test for available bandwidth and congestion. There are 3 types of CC algorithms: delay-based (e.g. TCP Vegas), loss-based (e.g. TCP Cubic) and hybrid forms (BBR-v2) [4], [5]. Delay-based algorithms change their sending rate according to the delay of the connection, similarly loss-based change theirs in case of packet loss. [1], [5]

Every packet a sender transmits is acknowledged by the receiver with acknowledgement packets (ACK). The number of packets a sender is able to send unacknowledged in each round-trip time (RTT) is called the *congestion window (cwnd)*. Every CC has three phases: 1) slow start where the available bandwidth is estimated 2) a steady phase aka congestion avoidance during which we roughly stay at our calculated bandwidth limit and probe for more slowly; and 3) loss recovery where CC reacts to packet loss. [4], [5]

The value of the *slow start threshold (ssthresh)* determines the change from slow start to congestion avoidance. In case of a loss event it is usually changed according to $sstresh = \beta \cdot loss\_cwnd$ where β denotes the *Multiplicative Decrease Parameter* and *loss_cwnd* is the *cwnd* right before a loss event or timeout. The window growth function $g(\cdot)$ defines how TCP grows *cwnd* in the congestion avoidance state and it makes certain CC algorithms very recognizable, e.g. TCP Reno with linear growth or TCP Cubic with a cubic function to have a very sensitive growth around the *loss_cwnd* and a rapid one otherwise. [2], [4]

3. Congestion Control Identification

Table 1 lists a variety of CCI methods available to this date. Because reviewing every single one in detail would

TABLE 1: Congestion Control Identification Overview

Method	Approach	Description	Included TCPs	Year
On Inferring TCP Behaviour (TBIT) [6]	active	trace congestion window to a given order of events, differentiate the 5 tcp methods based on this	4 (Tahoe, Reno, NewReno, TCP without Retransmit)	2001
Identification of different TCP versions based on Cluster Analysis [7]	passive	collect packets, extract features of cwnd based on a RTT estimate, cluster these to identify two competing CC variants	any 2 competing out of 14 (Reno, Cubic, BIC, CTCP, HSTCP, H-TCP, TCP Hybla, Scalable, Illinois, YeAH, Vegas, Veno, Westwood)	2009
TCP Congestion Control Avoidance Algorithm Identification (CAAI) [2]	active	extracts multiplicative-decrease parameter and window growth function, use machine learning to counter network conditions	14 (Reno, CTCP, BIC, Cubic, HSTCP, HTCP, HYBLA, ILLINOIS, LP, STCP, VEGAS, VENO, WESTWOOD+, YEAH)	2014
Identification of TCP Congestion Control Algorithms from Unidirectional Packet Traces [8]	passive	uses unidirectional packet trace, algebraic approach, approximate the whole SEQ-number to Time function, plot derivatives to differentiate TCP Congestion Control	5 (RENO, CUBIC, Hamilton TCP, Vegas, Veno)	2018
The Great Internet TCP Congestion Control Census (Gordon) [9]	active	similar to CAAI, different cwnd estimation algorithm	13 (BBR, Cubic, NewReno, BIC, HTCP, Scalable, Illinois, CTCP, YeAH, Vegas, Veno, Westwood, HSTCP)	2019
DeePCCI: Deep Learning based Passive Congestion Control Identification [3]	passive	only metric is packet arrival time, machine learning based classification with additional TCP Pacing differentation to increase identification accuracy, tests only in testbed	paper focused on BBR, CUBIC, RENO but trainable on any variant	2019
Inspector Gadget: A Framework for Inferring TCP Congestion Control and Protocol Configurations [5]	active	similar to CAAI with optimizations, especially improved network environments with changing RTT, Window Emptying and Sequence Check optimizations	12 (BBR, Cubic, Reno, BIC, hstcp, htcp, illinois, scalable, vegas, veno, westwood, yeah)	2020

go way beyond the scope of this work we are going to look at the general procedure of a CCI method and later dive into specific examples in Section 3.1 (CAAI) and 3.2 (DeePCCI). CAAI is from 2014 and while not being the newest active approach (see IG [5] and Gordon [9]), it gives a good understanding of the methodology. It's also based on a very early active approach in form of TBIT [6] from 2001. DeePCCI was chosen for pioneering an unconvential approach: ignoring TCP mechanics altogether and focusing only on packet arrival time.

Every CCI method follows a rough draft. First we need a way to get to the packet trace of the host we are interested in. Then we define features which enable us to differentiate between CC algorithms. These features are extracted from the packet trace and saved into a datastructure. Last but not least we match this processed representation of a host to some prepared data of each CC to classify the target. For this general procedure this work borrows the terminologies of **Trace Gathering**, **Feature Extraction** and **Algorithm Classification** from CAAI [2].

CCI methods can be generally categorized in two ways: TCP domain-dependent vs. TCP domain-independent and active vs. passive approaches. The first distinction differentiates between CCI methods that require knowledge of TCP in their implementation as they differentiate CC variants on subtle differences and methods that do not need knowledge of TCPs inner workings. Most methods are TCP domain-dependent but DeePCCI is a pioneer for the latter approach. The second distinction can be described in the following: Active approaches directly open up connections to a host to gain knowledge on the used CC by requesting data, manipulating the communication and observing the behaviour on the other side. Passive approaches do not interact with the observed host in any way. A host can be evaluated based on packet traces only, thus rendering this approach passive. One big advantage with passive approaches is the ability to work on real-world traffic. Packet traces could be captured on vantage points of a network or the Internet and therefore allow to identify alot of hosts without having to actively contact each one of them. A smaller subdistinction of passive methods can be made when looking at bidirectional vs. unidirectional packet traces. Most methods use bidirectional packet traces but for example Kato et al. [8] focused on unidirectional traffic only in 2018. [3]

3.1. Congestion Avoidance Algorithm Identification (CAAI)

CAAI is an active CCI method that is able to distinguish 14 different TCP algorithms as table 1 shows. Specifically its design goals aim to identify most default and non-default TCP algorithms while being insensitive to the operating system, network conditions and TCP components other than congestion avoidance of a webserver.

It characterizes each TCP congestion avoidance algorithm by two features:

- Multiplicative Decrease Parameter β

- Window Growth function $g(\cdot)$

The context of these variables in TCP has been explained in Section 2.

The reasoning for these two variables lies in the fact that different TCP algorithms have different multiplicative decrease parameters and congestion window growth functions. For example RENO uses $\beta = 0.5$ and a linear growth function of $g(x, \mathit{loss\_cwnd}) = 0.5 \cdot \mathit{loss\_cwnd} + x$ where x denotes the number of elapsed RTTs in the congestion avoidance phase, while CUBIC uses $\beta = 0.7$ and a window growth function that not only depends on x but also on the duration of a RTT. [2]

Trace Gathering: In step one CAAI gathers the TCP congestion window trace of a target in two simulated network environments using Linux' netem [10] to introduce various network conditions. To be able to differentiate 14 different TCP versions P. Yang et al. argue one needs different network environments in terms of the RTT. They settled for two network environments, A and B, and defined Environment A with a static RTT of $1.0s$ and Environment B with a RTT of either $0.8s$ or $1.0s$ to more easily distinguish RTT based *cwnd* changes e.g. CUBIC. These simulated environments depend on two more variables: *cwnd_threshold*, which sets the boundary for the timeout to start and should be high to help distinguish TCPs, and *mss*, which sets the maximum TCP segment size of the connection and should be low to enable a higher maximum *cwnd* value. For detailed reasoning of specific values like the chosen RTT or minor problems like Slow Start Treshold Caching on the webservers refer to [2]. The data transmission itself is initiated with repeated HTTP requests in form of HTTP pipelining and a tool developed by the authors to search for the longest webpage of a webserver to give a long enough transmission.

As in Section 2 explained, *cwnd* equals to the number of packets sent in one RTT. CAAI uses this constellation to estimate the *cwnd* of the target. Packets can be assigned to a specific RTT as the RTT value is high enough to have a bandwidth-delay-product much larger than *mss* $\cdot$ *cwnd_treshold*. This leads to the packet trace having lots of packets at the start of each RTT and then a gap to the next one. Further CAAI uses the highest received sequence number in one emulated RTT to counter lost packets that would impact the $cwnd$ estimation. [2]

Feature Extraction: To extract the two features from the trace CAAI first determines the boundary RTT that marks the change from slow start to congestion avoidance. It then extracts β with $\beta = w_s / loss\_cwnd$ where w_s is the congestion window size at the boundary RTT and $loss\_cwnd$ denotes the window size right before the timeout. The ssthresh formula from section 2 was essentially solved after β. The second feature, window growth function $g(\cdot)$, is extracted from the congestion window sizes after the boundary RTT using two measuring points $w_{s+4} - w_{s+1}$ and $w_{s+9} - w_{s+1}$. The substraction allows values independent of $loss\_cwnd$, e.g. RENO would have a value of $w_{s+4} - w_{s+1} = 3$ as it features a linear growth. Two points are enough to distinguish the every CC, Yang et al. argue. The values from both environments are then saved into one feature vector.

Algorithm Classification: The problem with the gathered data is its dependence on the congestion in the network at the time of the trace gathering. As a countermeasure CAAI employs a machine learning algorithm trained on the feature vectors of the 14 different TCP variants in different network conditions. CAAI uses random forest in this regard as it achieved the highest classification accuracy among the tested methods. [2]

3.2. DeePCCI

DeepCCI is a passive and TCP domain-independent approach. Existing CCI methods have weaknesses like 1) complexity when adding new CC algorithms as detailed knowledge about parameters and configuration are needed, 2) assumptions of missing extern influences like TCP pacing or static parameters, and 3) reliablity on parsable TCP header information. One example outside of TCP itself for problem 1) and 3) could be QUIC which moves CC to the userspace and implements fully encrypted transports. To counteract assumptions like these Sander et al. developed DeePCCI which uses packet arrival time as its only feature to distinguish CC variants and therefore stays flexible. DeePCCI uses the packet arrival time as any CC algorithm controls the packet flow in terms of amount and timing Sander et al. argue. [3]

Packet traces do not need to be gathered by the tool itself in passive methods. The packets in the trace are sorted into same-sized bins according to their arrival time to build a histogram $X = [x_0, ..., x_t]$ of packet arrivals with equidistant timesteps. This histogram is then fed into a deep neural network consisting of a convolutional neural network (CNN) and a long short-term memory (LSTM) part. The former is regarded as the feature extraction phase while the latter builds up memory depending on previous behaviour and is needed to identify varying length traffic flows as they appear in the real world. After the LSTM layer the neural network predictions are applied for CC classification and classification whether TCP pacing was present in the trace to help in CC classification.

The testbed to train the neural network consists of two main topologies with different network conditions in terms of amount of TCP senders, link latency, bottleneck link bandwidth and bottleneck queue sizes. A bottleneck link is central for changing these variables. Topology 1 is a single-host network with one TCP sender connected to a router which is connected via a bottleneck link to another router followed by the receiving host. Topology 2 is a little more complex with 3 hosts on each side. The training data in the latter topology consists of all possible combinations of the 3 CC algorithms (RENO, CUBIC, BBRv1) that DeePCCI focused on. In each setting traffic is captured before and after the bottleneck link. If other senders exist they start sending 2 s prior to the sender we are interested in and it sends traffic for 60 s. [3]

The previously mentioned variables and other measurements decisions have an impact on the distinction of CC which will be discussed now. **Bandwidth**: DeePCCI was able to distinguish delay-based (BBRv1) and loss-based CC very well for bandwidths above 10 Mbps. Bandwidths $\geq$ 10 Mbps and delays $\geq$ 5 ms resulted in F1 scores above 90% in the multi-host scenarios and with F1 scores above 55% in the more unrealistic scenario with one TCP sender. By including smaller bandwidths also, the minimum F1 scores drop to 55% (multi-host) and 40% (single-host) respectively. As a general trend we see that larger bandwidths, larger delays and multiple simultaneous traffic flows are beneficial for the result.

A higher bandwidth results in a higher maximum congestion window and therefore more steps of e.g. CUBIC are executed leading to an easier distinction from linear behavior as with RENO. **Delay**: One effect of the delay can be accounted to the bin size. A low delay makes it harder to distinguish CC variants as the change in packet arrival time would be too fast for a distinguishable difference in the histogram bin sub-sampling. The CCs would have a less unique histogram composition. It also influences the decision whether TCP pacing was used, with a low delay too many packets fall into the same bin, with or without pacing, impacting the decision negatively. Lastly, the multi-host scenario increases the queuing delays and leads to more congestion and therefore a competition for packets in the queue. This effect counteracts the effect of a low bandwidth and delay as 1) the delay increases with competing flow and 2) competing flows influence the *cwnd* of our target host, identifying his CC easier as we are not that dependant on alot of steps as explained before. [3]

3.3. Inspector Gadget

Inspector Gadget (IG) is a more recent work from 2020 that aims to identify a whole webserver's network stack configuration ranging from new default values like initial window size to whole new CC protocols. Its approach is active, TCP domain-dependent and it evaluates self-captured, bidirectional packet traces. The work additionally surveyed individual network operators from six distinct content delivery networks to find out about their approach to tuning their network stacks and the root cause of configuration heterogenity. Also the tool itself was tested on the Alexa top 5k websites and the work discussed TCP related anomalies in form of a measurement study of different Linux implementations in the wild. While the work of IG covers alot of topics, in the context of this paper we are mostly interested in CCI itself which is covered in IV.B: *Behaviour Parser Module*.

To seperate IG from the two previously shown methods we first look at the differences. Unlike DeePCCI with 3 CC algorithms, IG originally supports a broader range of algorithms. In contrast to CAAI it's also interoperatable with TLS/SSL, offers more optimizations to tackle domain-specific problems like pacing and puts an emphasis on delay variations to fingerprint delay-based CC algorithms. [5]

Similar to CAAI IG manipulates the RTT through delaying ACKs, enabling a set RTT of 0.8 s. As mentioned before it is also varied to fingerprint delay-based CC. To inject loss-events IG, just like CCAI, provokes timeouts. The *cwnd* estimation bears the first bigger change. Estimating the window by counting the packets received in

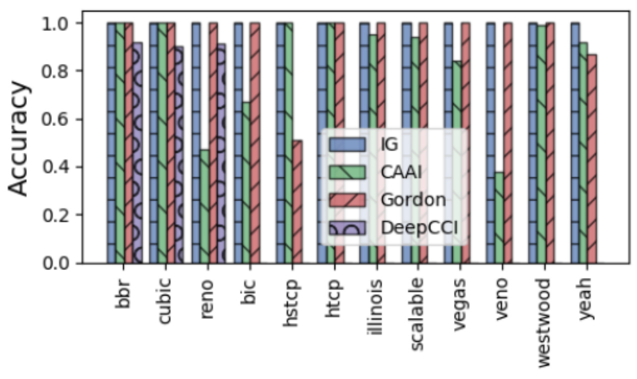


Figure 1: from [5]; Comparison of CAAI, DeepCCI, Inspector Gadget and Gordon

one RTT is prone to errors as this assumes that TCP is synchronous and ordered. Synchronous in the sense that at the start of a RTT the packets are sent in batch and the same for the ACKS at the end of a RTT. Also in the real world it can be observed that packets from one *cwnd* are spread over multiple RTTs due to e.g. pacing. Packet duplication and loss further worsen this scenario. To capture the *cwnd* accurately IG deploys two optimizations called *Sequence Check (SC)* and *Window Emptying (WE)*.

Packet reordering and duplication is a problem as the amount of packets in-flight differs from the actual congestion window. So instead of simply counting the number of packets in-flight IG uses SC to account for the TCP sequence number to identify and eliminate these cases.

Reasons like TCP pacing could also lead to a difference between packets in-flight and the real congestion window. Through the WE optimization ACKs are stored and sent batched at the end of a RTT. This ensures an empty sender window when sending the ACKs and a more accurate representation of the CC.

The *cwnd* trace is stored into the vector ν with each phase (Slow Start, Loss Recovery, Congestion Avoidance) seperated. Similar to Congestion Avoidance Algorithm Identification (CAAI) the values are saved as an offset, in this case from the first *cwnd* of each phase. For classification a decision tree with CART algorithm classifier was chosen. [5]

$$\nu = (\sum_{i=1}^{l} W_i - W_1, \sum_{i=1}^{x} W_{l+i} - W_{l+1}, \sum_{i=1}^{y} W_{s+i} - W_{s+1}) \tag{1}$$

As we see it is quite similar to CAAI but offers some optimizations. How these are impacting the result will be evaluated in section 4.

4. Evaluation

To compare these works with each other, we will first look at the impact of the SC and WE optimization in IG. Then we will analyse a comparison of these three CCI methods and elaborate on the results and differences.

IG Optimizations. Excluding all improvements in IG yields an accuracy loss of 62% in the worst case. The WE optimization had the biggest impact. Upon removal Gong et al. observed a false-positive rate of up to 31%. As a reminder, WE is responsible for synchronizing each RTT through ACK batching. In its abscence the SC feature has a higher probability to disregard packets as being not in the current window. This inaccurate trace leads to an inaccurate classification. Meanwhile a missing SC optimization leads to a minor drop to 92% from 100%. [5]

Accuracy Comparison. Gong et al. compared IG directly to CAAI, DeePCCI and Gordon, which we did not cover in this work except shortly in the CCI overview. To make results comparable, they extended CAAI's unmaintained source code with support for HTTPS and added a bottleneck to their setup to enable traffic capture on both sides for DeePCCI. The comparison with DeePCCI is somewhat restricted as it came with only 3 pre-trained CCs in form of Cubic, Reno and BBR. The environment itself consists of a server with the 4 CCI methods and webservers in an AWS Cloud to provide realistic network dynamics. The Linux traffic control tool was used to emulate different network conditions and each CC was fingerprinted 30 times for each network condition up to at least 4000 packets.

The results can be seen in figure 1. IG shows almost perfect identification across the different CC algorithms.

The re-implementation of CAAI featured accuracies between 41% and 94%. Even more disappointing is the poor result with major CC variants, for example BBR or Cubic with accuracies of 78% and 64% respectively. The reason for this may lie in the fact that CAAI emulates only two network conditions. Gong et al. argue that these two are not enough to capture small differences among certain CC algorithms, for example between Veno and Reno. In the original work of Yang et al. CAAI reached accuracies of 96.98% in their testbed but suffered of 53% invalid traces in Internet tests measuring 63124 popular webservers [2]. This number exists in the fact that CAAI could 1) not find a long enough webpage on a webserver to keep the connection up or 2) a webserver only accepts one or few HTTP requests in the same TCP connection and thus leading to slow start determining most of the data transmission [2].

Section 3.2 presented a first glance at the accuracy of DeePCCI under different testbed variables (e.g. amount of hosts, delay). It provides good results with accuracies above 96% in networks with client and server within the same area. But once WAN is introduced, as is the case in usual connections to webservers on the Internet, the accuracy drops to 90–92% in the tests of Gong et al. The main reason for this limitation might be the training with testbed generated data. Further DeePCCI might need to be retrained for CC variants across different kernels as they slightly differ. [5]

5. Conclusion

This work shed a light on CC fingerprinting. It reiterated on CC with its most important mechanics, phases and variables, and not only gave an overview on CCI algorithms, their methodology and possible categorizations but also reviewed three concrete examples with a finishing comparison in terms of accuracy under realistic WAN conditions. Furthermore an overview of CCI methods in form of table 1 has been provided.

References

[1] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[2] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP Congestion Avoidance Algorithm Identification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1311–1324, 2014.

[3] C. Sander, J. Rueth, O. Hohlfeld, and K. Wehrle, "DeePCCI: Deep Learning-Based Passive Congestion Control Identification," pp. 37–43, 2019.

[4] M. Allman and V. Paxson, "RFC5681—TCP Congestion Control," *RFC*, no. 5681.

[5] S. Gong, U. Naseer, and T. A. Benson, "Inspector Gadget: A Framework for Inferring TCP Congestion Control Algorithms and Protocol Configurations." International Federation for Information Processing, 2020.

[6] J. Padhye and S. Floyd, "On Inferring TCP Behavior," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 287–298, Aug. 2001.

[7] J. Oshio, S. Ata, and I. Oka, "Identification of Different TCP Versions Based on Cluster Analysis," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, 2009, pp. 1–6.

[8] T. Kato, X. Yan, R. Yamamoto, and S. Ohzahata, "Identification of TCP Congestion Control Algorithms from Unidirectional Packet Traces," in *Proceedings of the 2nd International Conference on Telecommunications and Communication Engineering*, ser. ICTCE 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 22–27.

[9] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The Great Internet TCP Congestion Control Census," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019.

[10] "netem," https://wiki.linuxfoundation.org/networking/netem, accessed: 2021-08-03.

Analysis of Proof of Stake flavors with regards to The Scalability Trilemma

Paul Schaaf, Filip Rezabek*, Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge75sab@mytum.de, frezabek@net.in.tum.de, kinkelin@net.in.tum.de

Abstract—**Blockchains are a rapidly evolving area of research and experimentation. Since Bitcoin's introduction in 2008, different protocols have been proposed and implemented with the goal of improving on Bitcoin's core feature, the Proof of Work consensus mechanism. A critical area many of the newer mechanisms focus on is a reduction in energy usage, for example.**

This paper presents and compares different Proof of Stake (PoS) mechanisms – an increasingly popular alternative to Proof of Work – that have been developed in recent years. The focus of our comparison are the mechanisms' abilities to solve the Scalability Trilemma, that is, a consensus mechanism's ability to achieve decentralization, security, and scalability. We find that Unbonded PoS is the most decentralized mechanism but comes with vague security assumptions. Bonded PoS is more secure at the cost of decentralization. Lastly, Delegated PoS achieves scalability but suffers from low decentralization and security.

Index Terms—**blockchains, proof of stake, scalability trilemma**

1. Introduction

Any given blockchain is a complex distributed system and the result of a variety of design decisions. One of the most important considerations is a blockchain's consensus mechanism. In decentralized networks where each node stores all the state (because there is no central server), there is a need for a mechanism that allows all nodes to come to consensus on which state changes should be applied to the current state and in which order they should be applied. This is so that after a state change all nodes have saved the same state [1].

In 2008, Satoshi Nakamoto used the concept of Proof of Work (PoW) to create Bitcoin's consensus mechanism. This mechanism makes nodes compete to solve hash puzzles (which is called *mining*) for Bitcoin rewards and the right to propose a specific set of changes [2].

Since then, Bitcoin's consensus mechanism has been criticized, mainly for its large consumption of energy because nodes are incentivized to buy more machines as long as the rewards outweigh the energy costs [3].

In an attempt to improve on the PoW consensus mechanism, other mechanisms have been researched and implemented in other blockchains. Most notably, Proof of Stake (PoS), a mechanism that removes the advantage of owning more machines and instead directly uses capital. Instead of mining, it employs a randomness function to choose the next block proposer which favors those with more capital. While PoS does not increase the fairness of the rewards distribution, it does come with lower energy requirements [4].

It is the goal of this paper to provide answers to the questions of what the subcategories of PoS are and how they differ. In sections 2 and 3 respectively, we present relevant background knowledge and highlight related work. Equipped with this knowledge, we compare the different flavors in section 4. In particular, we compare Unbonded Proof of Stake (UPoS), Bonded Proof of Stake (BPoS), and Delegated Proof of Stake (DPoS). The focus of this comparison is the ability of each mechanism to solve the so-called *Scalability Trilemma*. The trilemma states that it is difficult for any mechanism to achieve scalability without sacrificing security or decentralization [5]. It is these three properties by which we compare the mechanisms. In section 5, we summarize our findings and point to potential areas for future work.

2. Background

This section provides a high-level overview of how a blockchain works, presents the Proof of Work and Proof of Stake consensus mechanisms, and introduces the Scalability Trilemma.

2.1. Structure and Participants in a Blockchain System

The two main parties in a blockchain system are the users – those that wish to send transactions – and the nodes – those that maintain the blockchain. The blockchain is a log that contains all the transactions that have been committed so far. The transactions are aggregated in so-called *blocks*. Each block has a reference to the previous block, forming a chain of blocks (thus forming a special form of a linked list). The blockchain is kept as a local copy by every node because there is no central server. The current state of the network can be computed by replaying all transactions in the blockchain. Thus, every transaction moves the blockchain from one state into a different one. Using the example of cryptocurrencies, this state might save users' money which can be sent to other users by submitting transactions [6, Chapter 1-2].

2.2. Consensus

In Section 2.1 we have established that all nodes need to save an identical copy of the blockchain. Thus, when users request to send transactions, all nodes need to agree on which of the requested transactions should be added to the chain next and in which order, that is, what the next block to be added to the chain should look like. Allowing nodes to reach consensus is the goal of a consensus mechanism. Blockchain consensus mechanisms need to function in adversarial environments, that is, those in which there are malicious nodes [6, Chapter 2]. In particular, these mechanisms should be *sybil-resistant*, that is, they must continue to work under the assumption that malicious actors can create nodes at no cost [7]. Hence, a simple direct democracy where each node represents one vote with the winner being elected to propose the next block does not work [6, Chapter 2].

Subsections 2.2.1 and 2.2.2 explore the PoW and PoS mechanisms which are sybil-resistant.

2.2.1. Proof of Work. Proof of Work was first introduced in 1993 to prevent service abuses in computer networks [8]. It allows someone to prove they have done some work with the verification of this proof being cheaper than the work itself. We use Bitcoin as an example to illustrate how PoW blockchains incorporate this concept. All nodes in the system hold an individual block of requested transactions that could be added to the chain next. Each node tries to find a number – the *nonce* – so that when added to its block, the block hashes to a value smaller than a value X set by the protocol. There is no efficient algorithm to do this, so a node can only try different numbers (this is the *Work* in Proof of Work). If a node finds such a number, it broadcasts the block. Each node that receives it verifies the PoW (which equates to confirming the block's hash is smaller than X, which can be done in constant time) and the transactions in the block (e.g. the sender of some money must have agreed to the transfer). If valid, a receiving node adds the block to its local chain and starts working on the next one. This process is called *mining*. The result of this mechanism is that in regular intervals a new node is chosen to propose the new block. Note that a node is only chosen implicitly by finding the nonce; there is no voting taking place.

A benefit of PoW is that adding additional nodes does not increase the likelihood of finding the nonce, only the computing power matters. Hence, the Proof of Work consensus mechanism is sybil-resistant. In addition, it follows that the more distributed the computer power is across nodes, the more decentralized - and thereby more resistant to attacks - the network is [6, Chapter 2-4].

2.2.2. Proof of Stake. Proof of Stake is designed to be an alternative for PoW and was first proposed in a forum post in 2011 [9]. It works by essentially simulating the mining process. While Bitcoin's Proof of Work was described as *"essentially one-CPU-one-vote"* in the original whitepaper [2] by Nakamoto, Proof of Stake maps one unit of currency to one vote. *Validators*, as they are commonly called in PoS chains, deposit the respective blockchain's currency (in blockchain jargon: they *stake* and their capital is *staked*) and one validator is then chosen by a function to propose the next block. While incorporating some form of randomness, this function is more likely to choose a validator with a higher stake (again, analogous to owning more machines in PoW). One advantage of PoS over PoW is that the amount of machines a node controls does not increase the likelihood of being chosen as the next block producer, leading to a drastically lower energy footprint (about two thousand times more energy efficient in some cases [10]) [11].

The Nothing at Stake Problem. PoS consensus mechanisms have to deal with one issue PoW mechanisms do not: the Nothing at Stake Problem.

In general, blockchains can fork, meaning that two valid blocks are proposed at the same time, turning the blockchain into a tree with the two leaves referencing the previous block. In Proof of Work, a node chooses one branch by devoting its computing power to finding the nonce for the next block that references that branch's leaf. It could also use 30% on one branch and 70% on the other. The main point is that a node cannot use more than 100% of its computing power. A fork in PoW does not alter a miner's ability to produce blocks because the resource securing the network (the computers) is outside the network. One branch will likely have more computing power supporting it and eventually all nodes switch to that branch [6, Page 209].

The story is different in Proof of Stake, however [12]. A validator's likelihood to be a block producer is influenced by their stake, that is, the amount of the blockchain's native currency that they have staked. This means that the resource securing the network is part of the blockchain itself. Hence, when a fork happens, this resource is duplicated. Each validator wants to participate in consensus on the branch of the fork that eventually becomes the main branch because the rewards for participating on the eventually abandoned branch will not be considered real. However, because their capital is duplicated, they do not have to choose like block producers in PoW do, they can just produce blocks on both branches. It is trivial to see that if every validator thinks like this, there will never be a main branch because all branches continue producing blocks. After all, from the viewpoint of a single validator, there is no penalty for acting like this, a penalty that would e.g. remove the validator's funds. As a result, the validator has *nothing at stake* that they could lose if they act like described above. The reason this is a problem is that if forks do not clear up after some time, no consensus is reached and users of the currency cannot be sure their transactions are final.

2.3. The Scalability Trilemma

The Scalability Trilemma states that it is incredibly difficult for a blockchain to be scalable while staying secure and decentralized [5]. Note that for none of these properties there exists a single metric and a single value for that metric to achieve the property. This means in some cases it might be difficult to determine if one chain is, for example, truly more decentralized than the other. However, when comparing approaches where the differences are big enough, employing the trilemma makes sense. With this in mind, it is worth briefly exploring the three properties of the trilemma and which metrics could be used to measure them.

2.3.1. Decentralization. The rationale for decentralization is that in a peer-to-peer protocol, allowing one actor to take sole control is likely not in the interest of the other nodes. Two values that are often looked at are the number of nodes in a system and how many of them are controlled by a single entity [5].

2.3.2. Security. This property is the most obvious one. Any serious approach towards establishing a cryptocurrency should come with safety guarantees. A network should result in (possibly probabilistically) final transactions and be resistant to attacks.

2.3.3. Scalability. The two metrics that are most important when describing a blockchain's ability to scale are throughput and confirmation latency. That is, how many transactions can be executed per second and how long do users have to wait until they can consider a transaction non-reversible [3].

3. Related Work

With Bitcoin having been invented only in 2008, 13 years ago, research specifically on blockchains is still in the early stages (although many ideas Bitcoin and other blockchains rely on come from well-researched distributed systems theory and cryptography).

In addition, in this industry, much of the research that has been done has not necessarily been published in academic journals but in blog posts or projects' whitepapers. A relevant example: Vitalik Buterin – co-founder of Ethereum, the 2nd biggest cryptocurrency by market cap – reasons for Proof of Stake and how it relates to Proof of Work, and presents other related concepts like the Nothing at Stake Problem on his blog [13].

With that said, some more formal research has also been carried out. Narayanan et al. provide a thorough introduction to Bitcoin and cryptocurrencies [6]. Lepore et al. compare PoW, PoS and Pure Proof of Stake^{TM} and provide a framework for future comparisons [14]. Nguyen et al. present differences between the consensus algorithms of specific protocols (as opposed to the broader approach we take in this paper) and analyse staking pools, a phenomenon on PoS blockchains similar to mining pools on the Bitcoin network [4].

4. Recent Advances in PoS

This section focuses on recent advances in PoS algorithms. It first considers different approaches to solving the Nothing at Stake problem. Then, it explores Delegated PoS, a mechanism to increase the scalability of Proof of Stake. It also shows how each of these adjustments moves a protocol along the three dimensions of the Scalability Trilemma.

4.1. Solving the Nothing at Stake Problem

This section highlights two mechanisms that aim to solve the Nothing at Stake Problem explained in section 2.2.2 and describes the effects both mechanisms have on the properties of the Scalability Trilemma.

4.1.1. Bonded Proof of Stake. One solution to the Nothing at Stake Problem is *Slashing* [15]. The idea behind it is that nodes should only be allowed to produce a block if they have something to lose. To this end, the protocol requires a node to agree that if someone can prove that the node produced a block on two different forks (or more generally, act maliciously), the node loses part of its staked capital.

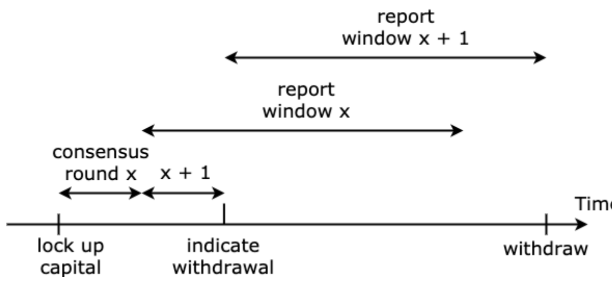


Figure 1: Example BPoS Participation Flow

Figure 2 illustrates this process, using an imaginary BPoS protocol. First, the participant locks up capital and begins participating in several consensus rounds - two rounds in this example (an unrealistically low number only chosen to conserve space). As shown in the figure, there is a report window for each consensus round in which others may report the node for malicious behaviour. This means that after the staker indicates their wish to withdraw, there is a period in which they have to wait and do not earn any rewards. Hence, their capital is *bonded*. Only once this period has passed, may they withdraw.

BPoS and the Scalability Trilemma. Bonded Proof of Stake is an addition to regular PoS that is meant to fend off the Nothing At Stake Problem. Hence, BPoS increases a blockchain's security.

However, BPoS introduces factors that might lead to centralization. First, the requirement to lock up capital limits the set of nodes to people that can afford to lock up their disposable income for such a purpose. Secondly, some PoS protocols allow users to delegate native currency (not to be confused with DPoS) to validators if they cannot or do not want to run their own node. This way, users can receive some of the maintenance rewards, while the validators take a fee. Generally, it is advised to split the delegations across multiple validators to maintain the decentralization of the network. In networks that employ slashing, however, users will think twice about whom they are delegating their money to. In practice, this has led to stake being concentrated across a few trusted companies in many PoS protocols [16], [17].

Bonded Proof of Stake is only meant to make a blockchain more resistant and has, in itself, no effects on scalability.

BPoS in Practice. The most prominent protocol using Bonded Proof of Stake is Ethereum 2.0 (ETH2) [18]. To become a validator, a deposit of 32 ETH (worth about $55,000 as of June 13, 2021 [19]) is required. If a validator produces a block on two forks, it is slashed whereas the amount slashed increases the more validators act maliciously. If a third of validators act maliciously at roughly the same time, their entire deposit is slashed [20], [21].

4.1.2. Unbonded Proof of Stake. Another way to approach the Nothing at Stake problem is to bypass it by using different assumptions. The most important one is that the average member of society is virtuous; in particular, they do not wish to hurt society or its monetary system. There may be parts of society that act maliciously but that part is small in any functioning society. Thus, assuming that the node distribution in a blockchain resembles society, most nodes will not produce blocks on two different forks or be bribed to do so because this would hurt the blockchain's health. Hence, there is no need to lock up capital and punish bad behavior because it will never happen on a scale large enough to affect the network anyway [22], [23].

UPoS and the Scalability Trilemma. UPoS moves the protocol using it further towards decentralization because there is no lockup of capital. However, it relies on more assumptions than Bonded Proof of Stake, mainly that the average member of a functional society is virtuous and that a blockchain's node distribution resembles society. Especially the latter assumption is debatable because in a public UPoS blockchain system, nodes are anonymous and there are no punishments, unlike most human societies. Open questions like these make it impossible to give a clear answer on the mechanism's security for now.

Analogous to BPoS, UPoS, in itself, does not affect scalability.

UPoS in Practice. Algorand is a blockchain using a mechanism called Pure Proof of Stake™ (PPoS) [22]–[24]. PPoS does not require any lockup of capital. However, available capital is still used to grant proportional voting power to nodes in the network. Each round, a chosen node's block is voted on by random committees of nodes. Using a subset of all nodes to add the next block increases efficiency. The random composition of these committees makes it highly likely that they resemble the overall network. To find a node's role for the creation of the next block (e.g. block proposer or committee member), it executes a local verifiable random function that requires no communication with other nodes. This can result in multiple nodes proposing a block. However, only the block with the highest priority (which it is more likely to have the more of the native currency the proposers owns) will be accepted by the committee. This results in nodes only having a single block to vote on which they do if it is valid and they are honest.

4.2. Delegated Proof of Stake

Delegated Proof of Stake is a consensus mechanism that combines PoS with a governance system [25]. Its goal is to increase a PoS blockchain's scalability. Its main innovation is that it only allows a fixed number of representative nodes to participate in consensus. With only a small number of nodes required to come to consensus, the communication overhead drops and consensus is reached faster.

DPoS consists of two steps. First, an election takes place to determine the representative nodes. All nodes can participate and they use the blockchain's native currency to vote. From the election onwards, the blockchain may use another PoS mechanism to let the elected nodes come to consensus for several blocks after which a new election takes place. Hence, this mechanism has similarities to a representative democracy.

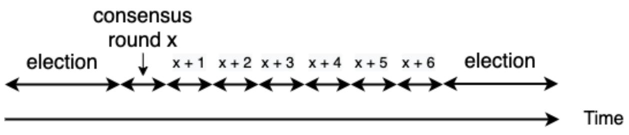


Figure 2: Example Delegated Proof of Stake

Figure 2 shows an imaginary DPoS protocol where an election happens every 7 rounds. Important to note, the election round takes longer than the consensus rounds because it has more participants.

DPoS and the Scalability Trilemma. DPoS provides a fixed and low number of consensus nodes, thereby increasing a blockchain's ability to scale. However, this benefit comes at the cost of decentralization and security. It is trivial that any system in which a small, fixed number of nodes have all the control is less decentralized than a system in which control is spread across an unlimited number of nodes. In addition, even if the elected nodes are not malicious, attacking a low, fixed number of nodes is easier than attacking a large number of nodes that are possibly not known in advance.

DPoS in practice. This paragraph briefly showcases the best known DPoS blockchain: EOS [26] (this is indeed its name, not an abbreviation). In EOS, a new block is produced by one of 21 consensus nodes every 0.5 seconds. Next to the 21 consensus nodes that produce blocks, there are about 530 other nodes on standby that could be elected as of June 15, 2021. The consensus nodes receive EOS (the EOS blockchain's native currency) as a reward for maintaining the system. The approximately 40 nodes on standby with the highest number of votes also receive rewards, the others do not. A new election for the consensus nodes happens every 126 blocks. Consensus nodes use the asynchronous Byzantine Fault Tolerance (aBFT) consensus algorithm which ensures transaction finality after 1 second. The maximum observed transactions per second to date on EOS is about 4000 [27]. This example demonstrates that EOS is indeed more scalable than, for instance, Bitcoin and Ethereum as of June 15, 2021 (with about 7 and 15 tps, respectively [28]). However, EOS has received some criticism regarding malicious behavior by its consensus nodes. In particular, allegations of vote buying (a consensus node bribes token holders to vote for it) and mutual voting (consensus nodes with large amounts of EOS agree to vote for each other to stay in power) have been made public [29], [30].

5. Conclusion and Future Work

In this paper, we have presented and compared different flavors of the Proof of Stake consensus mechanism with regards to their ability to solve the Scalability Trilemma.

In sections 1 and 2 we motivated the need for consensus mechanisms, which allow nodes in a distributed system to agree on state changes. We then presented two broad categories of consensus mechanisms, the Proof of Work mechanism – employing computing power to reach consensus – and the Proof of Stake mechanism – employing capital to reach consensus, thereby achieving a lower energy footprint.

In section 4 we compared flavors of the Proof of Stake consensus mechanism. The metrics of the comparison were the three properties of the Scalability Trilemma. The summarized findings can be found in table 1.

Mechanism	Scalability	Decentralization	Security
BPoS	0	-	+
UPoS	0	0	0
DPoS	+	-	-

TABLE 1: Comparison of PoS flavors,
0 $\hat{=}$ base case, + $\hat{=}$ increase, - $\hat{=}$ decrease

UPoS is used as the base case from which the other two are compared in the table because as we have shown it is the mechanism which is the easiest to achieve – it only requires a change of assumptions. We have found that BPoS alone does not offer any scalability improvements over UPoS whereas DPoS does, at the cost of both decentralization and security. Punishable capital requirements in BPoS increase security at the cost of decentralization. An important takeaway from this comparison is that none of the presented flavors can satisfy all three properties of the Scalability Trilemma.

At this point, it is important to mention, however, that these three flavors represent three sets of mechanisms with each set containing blockchains that still differ significantly. In addition, while the two sets of BPoS and UPoS mechanisms are mutually exclusive, BPoS/UPoS and DPoS are not. Hence,one should, for example, not conclude that a blockchain using UPoS cannot scale. A mechanism that is only in the UPoS set may still scale by employing other technologies but this increase in scalability is not caused by UPoS itself. One such example is Algorand, discussed in section 4.1.2, which increases scalability through short-lived stake-weighted random committees (as opposed to elected longer-lived ones in DPoS).

As a result, comparing these broad categories in a vacuum is only the tip of the iceberg. A more practical comparison in the future might compare how these mechanisms are used in conjunction with other technologies in different protocols. In addition, a more holistic analysis would also include innovations that are above the protocol level. In BPoS, for example, there exists the concept of Liquid Staking on the application layer. It is meant to alleviate the problems with locked-up capital by allowing stakers to withdraw staked capital immediately for a fee [31].

References

[1] "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," *IEEE Access*, vol. 7, pp. 22 328–22 370.

[2] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Accessed: 15/05/2021. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to Scalability of Blockchain: A Survey," *IEEE Access*, vol. 8, pp. 16 440 – 16 455, Jan. 2020.

[4] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019.

[5] A. Altarawneh, T. Herschberg, S. Medury, F. Kandah, and A. Skjellum, "Buterin's Scalability Trilemma viewed through a State-change-based Classification for Common Consensus Algorithms." IEEE, 2020.

[6] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[7] B. N. Levine, C. Shields, and N. Boris Margolin, "A Survey of Solutions to the Sybil Attack," 2005.

[8] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," E. F. Brickell, Ed. Springer-Verlag, 1992, p. 139–147.

[9] QuantumMechanic. Proof of stake instead of proof of work. Accessed: 15/05/2021. [Online]. Available: https://bitcointalk.org/index.php?topic=27787.0

[10] A country's worth of power, no more! [Online]. Available: https://blog.ethereum.org/2021/05/18/country-power-no-more/

[11] F. Saleh, "Blockchain Without Waste: Proof-of-Stake," *Review of Financial Studies*, vol. 34, pp. 1156–1190, 2021.

[12] V. Buterin. (2014) On Stake. Accessed: 15/05/2021. [Online]. Available: https://blog.ethereum.org/2014/07/05/stake/

[13] ——. Accessed: 06/06/2021. [Online]. Available: https://vitalik.ca/

[14] C. Lepore, M. Ceria, A. Visconti, U. P. Rao, K. A. Shah, and L. Zanolini, "A Survey on Blockchain Consensus with a Performance Comparison of PoW, PoS and Pure PoS," *mathematics*, vol. 8, 2020.

[15] V. Buterin. (2014) Slasher: A Punitive Proof-of-Stake Algorithm. Accessed: 15/05/2021. [Online]. Available: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[16] Cosmos Validators Leaderboard. Accessed: 13/06/2021. [Online]. Available: https://web.archive.org/web/20210613103151if_/https://cosmos.fish/leaderboard/all

[17] Eth1 Deposit Addresses. Accessed: 13/06/2021. [Online]. Available: https://web.archive.org/web/20210613103156/https://beaconcha.in/charts/deposits_distribution

[18] eth2. Accessed: 15/05/2021. [Online]. Available: https://ethereum.org/en/eth2/

[19] Ethereum USD Historical Data. [Online]. Available: https://www.coingecko.com/en/coins/ethereum/historical_data/usd#panel

[20] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," 2017.

[21] Serenity design rationale. Accessed: 13/06/2021. [Online]. Available: https://web.archive.org/web/20210613112919/https://notes.ethereum.org/@vbuterin/rkhCgQteN#Slashing-and-anti-correlation-penalties

[22] Various questions about the algorand blockchain. Accessed: 05/06/2021. [Online]. Available: https://medium.com/algorand/various-questions-about-the-algorand-blockchain-ef8bf719f1f

[23] Silvio micali's lecture on algorand. Accessed: 05/06/2021. [Online]. Available: https://youtu.be/NykZ-ZSKkxM

[24] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies." Association for Computing Machinery, 2017, pp. 51–68.

[25] Z. Zheng, H.-N. Dai, and S. Xie, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, Jan. 2018.

[26] EOS Consensus Protocol. Accessed: 15/05/2021. [Online]. Available: https://developers.eos.io/welcome/v2.0/protocol/consensus_protocol

[27] Bloks.io | fastest eos block explorer and wallet. Accessed: 15/05/2021. [Online]. Available: https://www.bloks.io/

[28] Sharding FAQs. [Online]. Available: https://eth.wiki/sharding/Sharding-FAQs

[29] Corrupt governance? what we know about recent eos scandal. Accessed: 15/05/2021. [Online]. Available: https://cointelegraph.com/news/corrupt-governance-what-we-know-about-recent-eos-scandal

[30] V. Buterin. Governance, part 2: Plutocracy is still bad. Accessed: 15/05/2021. [Online]. Available: https://vitalik.ca/general/2018/03/28/plutocracy.html

[31] M. Di Maggio. Liquid staking: A discussion of its risks and benefits. Accessed: 06/06/2021. [Online]. Available: https://web.archive.org/web/20210606192846/https://medium.com/terra-money/liquid-staking-a-discussion-of-its-risks-and-benefits-bbaa957d9233

Survey on Back-Pressure Based Routing

Ke Wang, Christoph Schwarzenberg*, Florian Wiedner*,
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: 19.ke.wang@tum.de, schwarzenberg@net.in.tum.de, wiedner@net.in.tum.de

Abstract—**Back-pressure based routing-algorithms have been studied extensively. They guarantee throughput optimality, but have poor delay performance. In this paper, we focus on wired Ethernet networks and compile a survey of back-pressure based routing-algorithms. Four variants are presented in this paper. We address their advantages, limitations, and possible combinations of different variants.**

Index Terms—**back-pressure routing, shortest-path, delay metric, cluster, machine learning**

1. Introduction

The back-pressure routing algorithm (BP algorithm) first introduced by L. Tassiulas and A. Ephremides in [1] draws many researchers' attention and has been studied extensively since then. Although it was initially proposed for wireless multi-hop radio networks, it can be applied to wire-line networks. The algorithm directs packets in multi-hop queuing networks based on congestion gradients. By exploring all possible routes to balance the loads, it guarantees network-wide throughput optimality [1].

However, there are still plenty of problems that need to be solved. As stated in [2], the BP algorithm requires routers to maintain a separate queue for each destination, which results in high memory complexity. Another problem that hinders the deployment of the algorithm is the poor delay performance. The algorithm works well only with heavy traffic loads. When the traffic loads are light or moderate, BP may lead to packets being directed to unnecessarily long routes or even loops [3].

Since BP guarantees throughput optimality, it has great potential to be applied in the industry. Therefore, many variants have been proposed to solve the delay and memory consumption issues.

2. Related work

In [4], Ying et al. use the shortest path method to avoid the extensive exploration of paths. In [5], Anurag Rai et al. propose to use directed acyclic graphs to eliminate loops in the network, which in turn improves the delay performance.

In [2], [6], [7], queue structure and management are improved to reduce the packet delay. In [6], Alresaini et al. introduce an adaptive redundancy technique that yields the benefits of replication, while at the same time preserving the benefits of traditional BP routing algorithm under high traffic loads. In [7], Ji et al. design a new queue management policy with a delay parameter. The algorithm will then select favorable routes by considering both delay requirements and network throughput. In [8] Bui et al. design a shadow queuing architecture that improves the delay performance for the original BP algorithm. In [9] Athanasopoulou et al. combine BP algorithm with probabilistic routing tables and shadow queues to decouple routing and scheduling in the network. In [10] Moeller et al. combine the BP algorithm with the LIFO queuing discipline. In [11] Huang et al. prove that the algorithm achieves near-optimal utility-delay trade-off. In [12] Gao et al. provide a general framework by combining several parameters addressed above to reduce the delay.

Another approach for improving the algorithm are delay-based BP algorithms [13]–[16]. In [13], Hai et al. propose a novel delay metric called sojourn time backlog and improve the BP algorithm by using this metric instead of backlog. In [16], Mekkittikul et al. propose a delay-based approach that uses head-of-line delays instead of queue lengths. Michael J. Neely analyses the approach with Lyapunov optimization for one-hop wireless networks in [14]. Ji et al. extend the approach to multi-hop wireless networks [15].

With the great progress in the field of data science, machine learning provides another solution for improving the original BP algorithm. In [17], Huang et al. investigate the benefit of predictive scheduling and establish a novel queue-equivalence result based on a look-ahead prediction window model. In [18], Gao et al. combine the traditional BP algorithm with multi-agent Q-Learning, and have shown that it reduces the average packet delay by 95% for light traffic loads.

3. Variants of the BP Algorithm

To improve the delay performance and reduce the memory complexity of the original BP algorithm, we compare four variants in this section, and present their method of operation, advantages, and disadvantages.

3.1. Reduce the Path Length

In [4], Ying et al. aim to minimize the average number of hops per packet delivery, or the average path lengths between sources and destinations. It has two interpretations. First, the number of hops can be thought of as the number of transmissions needed to support traffic. Minimizing it can be regarded as minimizing the network resource. Second, the number of hops is related to end-to-end delay. Decreasing the number of hops results in a lower delay.

They propose a joint traffic-control and shortest-path-aided BP algorithm. When the traffic is light, the algorithm chooses the shortest paths; when the traffic increases, more paths are exploited to support the traffic. To control the trade-off between shortest path selection and back pressure policy, a tuning parameter K is used. When K becomes quite large, the algorithm only uses the optimal path. The optimal value of K depends on the networks. The strategy does not only guarantee network stability (throughput-optimal), but also adaptively selects the optimal path according to the traffic demand [4].

To study the performance of the algorithm, the authors implement the simulation using OMNeT++. The setup is shown in [4]. All intercluster flows have the same arrival rate denoted by λ (packets/time slot). λ is used to observe the performance of the algorithm under different traffic loads. The performance of shortest-path based BP with different K is shown in Figure 1. A small K results in a small penalty on long paths. For $K = 100$, the penalty on long paths is too large, therefore the algorithm will only prefer the shorter path without considering the backlog length.

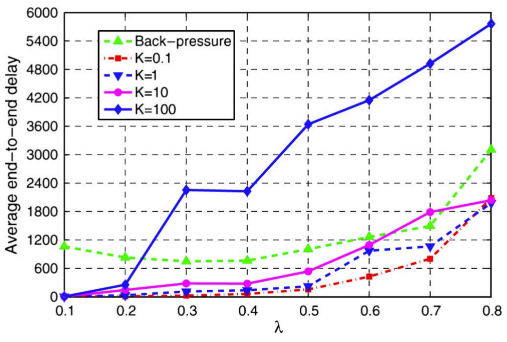


Figure 1: Simulation Result From [4]

The algorithm is simple and the performance is improved compared to traditional BP. It requires the calculation of the multiple source multiple destination shortest path, which can be calculated by Floyd-Warshall in $\mathcal{O}(N^3)$. For each node in the network, they need to store the path information besides the backlog. A pretty similar approach is also introduced in [9], [19].

Another aspect to reduce the path length is to remove loops before applying the BP algorithm as shown in [5]. The authors propose to assign directions to the links so that the network becomes a directed acyclic graph (DAG). Initially, an arbitrary DAG is generated and then the BP algorithm is used. When certain links are overloaded, a new DAG is created by reversing the direction of the links that point from non-overloaded to overloaded nodes. This approach avoids loops, thus the end-to-end delay is decreased. By iteratively creating new DAGs and performing BP, the throughput optimality is guaranteed.

3.2. Cluster the Nodes

In [2], Ying et al. show that the end-to-end delay is decreased without loss of throughput by properly clustering the nodes. The criteria for clustering nodes, e.g. geometrically by the frequency of information exchange, does heavily depend on the network. After clustering, routers need to maintain one queue for each cluster, therefore the variant also reduces the memory complexity significantly compared to the original BP algorithm. The principle is similar to the routing algorithm between different autonomous systems (AS). Packets are sent along the gateway between different clusters.

The cluster-based BP algorithm consists of three components, i.e. **traffic controller, regulator, and back-pressure scheduler**. The traffic controller decides the least congested gateway and how many packets can go through the gateway. The regulator is for limiting how many packets can be transferred through a certain gateway. The back-pressure scheduler decides the best route for transferring the packets.

The authors simulate the algorithm using OMNeT++. The setup is shown in [2]. The simulation result is shown in Figure 2. *Cluster-bp-w/o* and *cluster-bp-w* are two variants of cluster based BP. Both of them are better than the traditional BP according to the figure. Another interesting fact is the dramatic increase of delay of the shortest path algorithm when λ is equal to 0.5. This is because the algorithm only chooses shortest paths. When all paths are fully loaded and the traffic still increases, the delay grows significantly. The authors also propose further variants of cluster-based BP algorithm, such as multilevel clustering and combining it with policy-based routing.

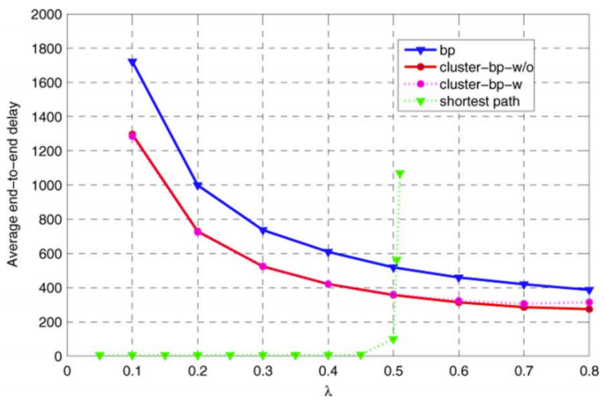


Figure 2: Simulation Result From [2]

In [20], the authors introduce a new metric called greedy back-pressure metric value (GBM). GBM values are evaluated to route the packets toward gateways in the direction of the steepest gradient. It uses a combination of traffic load and the mesh node's hop count to the nearest gateway. The authors show that the GBM based back-pressure algorithm outperforms the traditional BP algorithm. Cluster-based BP can be improved by combining the GBM method, since the GBM method utilizes the gateways more efficiently.

Even though the simulation of [2] gives a decent result, one important problem is how to cluster the nodes properly. The authors give some suggestions such as clustering nodes according to the network topology or physical location. But it is still an open question that needs to be solved.

3.3. Delay Based Algorithm

In [13], the authors introduce a new delay metric called the sojourn time backlog (STB). The STB considers the queue length and accumulated packet delays comprehensively. The authors propose an STB-based back-pressure algorithm called STBP. It applies the BP algorithm based on the STB. STBP routes the packets to a shorter or faster path compared to the traditional BP algorithm. The authors prove that STBP is stable and throughput optimal.

One challenge of implementing STB is to realize time synchronization (a well-known challenge [21]). For packets moving between different nodes in the network, the clocks of these nodes need to be synchronized so that the sojourn time is correctly recorded. The accuracy of the synchronization affects the performance significantly. To overcome the problem, the authors propose to use hop-count instead of exact sojourn time.

To illustrate the performance, the authors simulate the algorithm using the NS-2 network simulator. They compared STBP and STBP-hop based algorithm with the traditional BP algorithm. The setup is described in [13]. The performance is shown in Figure 3. Before the saturated point, all algorithms have similar performance except STBP. After 100 kb/s, STBP and STBP-hop have smaller delays compared to traditional BP. Another fact is that the hop-count (STBP-hop) method performs worse than the STBP method based on the figure.

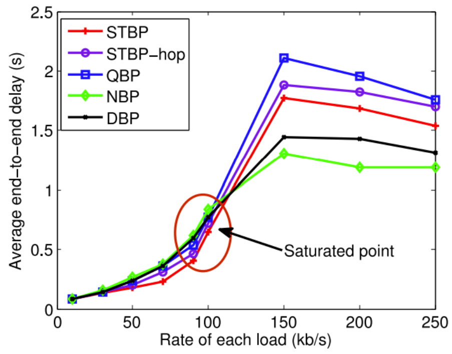


Figure 3: Result From [13], traditional BP is denoted as QBP

In [10], the authors combine the traditional BP algorithm with the LIFO policy. The algorithm transfers the new packets to their destination with less waiting time. In [11] the authors improve the algorithm further by combining the LIFO and the FIFO policy. At every time slot, the algorithm randomly decides to serve packets from either the back of the queue or the front of the queue. It avoids some packets staying in the queue for long time. The authors prove that the algorithm achieves close-to-optimal performance and decreases the delay. This kind of strategy is close to the STBP, which aims to reduce the waiting time of the packets. It is easier to implement and does not require synchronization between different nodes. But it does not always produce the optimal result.

3.4. Combine with Data Science

The fast development of data science draws the attention of researchers in many areas. In [17], [18], the authors combine machine learning with the traditional BP algorithm.

In [17], the authors propose a *lookahead window model* to pre-allocate rates. The lookahead window, also called prediction queue, is constructed by the server according to the previous packets. The lookahead window helps the server to use links more efficiently. They perform the BP algorithm based on the prediction queue. The authors prove that the algorithm achieves a cost performance that is arbitrarily close to the optimality, while guaranteeing that the average system delay vanishes as the prediction window size increases. The reason is that with larger window size, the prediction is more accurate. They also simulate the algorithm in a 10-user single server system. The result shows that when the prediction window size increases, the network delay decreases. However, the algorithm requires more computational power and more time to work properly.

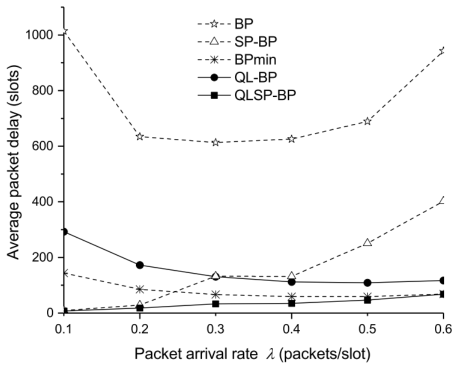


Figure 4: Simulation Result from [18]

In [18], the authors use multi-agent Q-learning to extract biases and based on these biases to perform the BP algorithm. Q-learning is a variant of reinforcement learning, it keeps learning based on the environment according to the reward [22]. Each node in the network estimates route congestion using local information of the neighboring nodes. And every node has multiple Q-learning agents that continuously update its route congestion estimate. Based on the estimate, every node then directs packets via the least congested routes to their destinations.

The algorithm is based on the *Bias Based General Framework* [12]. The framework consists of three parts, i.e. *information collection, bias extraction, and back-pressure routing*. At the information collection stage, the framework collects useful information (local or global) such as queue length, shortest path, and packet delay, for delay reduction. As for bias extraction, the framework extracts useful biases. Finally, the BP algorithm directs packets based on the features from the second stage.

The authors prove that the Q-learning based BP algorithm is throughput optimal. They also simulate the algorithm to test its performance, the result is shown in Figure 4. The traditional BP algorithm is denoted as BP, and QL-BP is the abbreviation for the Q-learning aided back-pressure algorithm. QL-BP's delay performance is much better than original BP.

Since each node extracts the bias based on its local knowledge, this enables distributed implementation. The

TABLE 1: Overview of all variants

Variant	Improvement	Challenge
Reduce path length	Shortest path, Remove loop, LIFO	Balance delay and throughput
Cluster nodes	Cluster nodes using gateway	How to cluster
Delay based algorithm	Delay metric	Time synchronization
Data science	Prediction	Computation, Distribution

computation complexity is low compared to algorithms maximizing the weighted sum globally. Even though it requires computation for extracting biases, but with the growth of the computational power of the electronic components, it provides a good way for improving the back-pressure algorithm.

4. Summary and Conclusion

We have introduced four variants of BP based algorithms. Table 1 shows the overview of all variants. All methods reduce the delay compared to the traditional BP algorithm. The Cluster-based algorithm reduces the memory complexity. With the rapid development of data science, machine learning has potential to decrease the delay further.

There are still challenges. The balance between delay and throughput is an important problem for path-related algorithms. As for clustering, how to cluster nodes in the network properly is still an open question, and it is also a fundamental requirement for applying the cluster-based algorithm. To use the delay-based algorithm, it is crucial to synchronize the time. Otherwise, the nodes are not able to record the delay of the packets properly and thus the algorithm cannot work. For machine learning methods, it requires more computation time, and nodes need to constantly record the data and update the parameters.

In this paper, we discussed the advantages and disadvantages of every variant. Reducing the path length can be achieved by the shortest path algorithm, removing loops in networks, using DAGs or LIFO. This variant reduces the delay when the traffic load is light. Clustering the nodes properly can reduce the memory complexity and end-to-end delay. The delay-based algorithm reduces the delay while guaranteeing throughput optimality. The last variant is machine learning. Machine learning helps to route the packets efficiently with near throughput optimality. We also proposed potential combinations of different variants, such as combining clustering method with GBM values.

References

[1] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1949, 1952.

[2] Ying, Lei and Srikant, R. and Towsley, Don and Liu, Shihuan, "Cluster-Based Back-Pressure Routing Algorithm," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1773–1786, 2011.

[3] Gao, Juntao and Shen, Yulong and Ito, Minoru and Shiratori, Norio, "Bias Based General Framework for Delay Reduction in Backpressure Routing Algorithm," *2018 International Conference on Computing, Networking and Communications (ICNC)*, pp. 215–219, 2018.

[4] Ying, L. and Shakkottai, S. and Reddy, A., "On Combining Shortest-Path and Back-Pressure Routing Over Multihop Wireless Networks," *IEEE INFOCOM 2009*, pp. 1674–1682, 2009.

[5] A. Rai and C. Li and G. Paschos and E. Modiano, "Loop-Free Backpressure Routing Using Link-Reversal Algorithms," *IEEE/ACM Transactions on Networking*, vol. 25, no. 05, pp. 2988–3002, 2017.

[6] Alresaini, Majed and Sathiamoorthy, Maheswaran and Krishnamachari, Bhaskar and Neely, Michael J., "Backpressure with Adaptive Redundancy (BWAR)," *2012 Proceedings IEEE INFOCOM*, pp. 2300–2308, 2012.

[7] Ji, Zhe and Wang, Youzheng and Lu, Jianhua, "Distributed Delay-Aware Resource Control and Scheduling in Multihop Wireless Networks," *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, pp. 1–5, 2015.

[8] Bui, Loc X. and Srikant, R. and Stolyar, Alexander, "A Novel Architecture for Reduction of Delay and Queueing Structure Complexity in the Back-Pressure Algorithm," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1597–1609, 2011.

[9] Athanasopoulou, Eleftheria and Bui, Loc X. and Ji, Tianxiong and Srikant, R. and Stolyar, Alexander, "Back-Pressure-Based Packet-by-Packet Adaptive Routing in Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 244–257, 2013.

[10] Moeller, Scott and Sridharan, Avinash and Krishnamachari, Bhaskar and Gnawali, Omprakash, "Routing without Routes: The Backpressure Collection Protocol," *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, p. 279–290, 2010.

[11] Huang, Longbo and Moeller, Scott and Neely, Michael J. and Krishnamachari, Bhaskar, "LIFO-Backpressure Achieves Near-Optimal Utility-Delay Tradeoff," *IEEE/ACM Transactions on Networking*, vol. 21, no. 3, pp. 831–844, 2013.

[12] Gao, Juntao and Shen, Yulong and Ito, Minoru and Shiratori, Norio, "Bias Based General Framework for Delay Reduction in Backpressure Routing Algorithm," *2018 International Conference on Computing, Networking and Communications (ICNC)*, pp. 215–219, 2018.

[13] Hai, Long and Gao, Qinghua and Wang, Jie and Zhuang, He and Wang, Ping, "Delay-Optimal Back-Pressure Routing Algorithm for Multihop Wireless Networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 2617–2630, 2018.

[14] Neely, Michael J., "Delay-Based Network Utility Maximization," *2010 Proceedings IEEE INFOCOM*, pp. 1–9, 2010.

[15] Ji, Bo and Joo, Changhee and Shroff, Ness B., "Delay-Based Back-Pressure Scheduling in Multihop Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1539–1552, 2013.

[16] McKeown, N. and Mekkittikul, A. and Anantharam, V. and Walrand, J., "Achieving 100% Throughput in An Input-queued Switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[17] Huang, Longbo and Zhang, Shaoquan and Chen, Minghua and Liu, Xin, "When Backpressure Meets Predictive Scheduling," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2237–2250, 2016.

[18] Juntao Gao and Yulong Shen and Minoru Ito and Norio Shiratori, "Multi-Agent Q-Learning Aided Backpressure Routing Algorithm for Delay Reduction," *CoRR*, vol. abs/1708.06926, 2017.

[19] Yin, Ping and Yang, Sen and Xu, Jun and Dai, Jim and Lin, Bill, "Improving Backpressure-based Adaptive Routing via Incremental Expansion of Routing Choices," *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1–12, 2017.

[20] Hu, Bin and Gharavi, Hamid, "Greedy Backpressure Routing for Smart Grid Sensor Networks," *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pp. 32–37, 2014.

[21] "Time Synchronization," https://www.cs.usfca.edu/~srollins/courses/cs686-f08/web/notes/timesync.html.

[22] "An introduction to q-learning: Reinforcement learning," https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/.

An Implementation of the Babel Routing Protocol for ns-3

Malte von Ehren, Jonas Andre*, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: malte.von.ehren@tum.de, andre@net.in.tum.de, wiedner@net.in.tum.de

***Abstract*—This paper introduces an implementation of the Babel routing protocol for the discrete event simulator ns-3. Babel is a relatively new general-purpose routing protocol with no previously existing implementation in ns-3. This paper motivates the need for network simulation in general as well as Babel in particular. We also outline implementation details and validate the implementation by simulating network setups with Babel in ns-3 and comparing them with the expected behavior.**

***Index Terms*—Routing protocols, Routing, Babel Routing Protocol, ns-3, network simulation**

1. Introduction and Related Work

Babel is a robust, general routing protocol, well suited for both wired and wireless networks. It addresses the shortcomings of other routing protocols for wireless mesh routing. Its design makes it loop avoidant and claims to have faster convergence than similar protocols [1].

Network simulation is an essential tool in network research as evaluating routing performance and comparing different routing setups in real-world tests is time-consuming and expensive. Network simulation helps to investigate the most important properties and can test a variety of parameters quickly. In the context of routing protocols, it is, for example, possible to evaluate how different parameters impact bandwidth usage and convergence time of a routing protocol. Network simulation is also helpful to familiarize oneself with the operation of a protocol since one can look at the traces of involved nodes with little effort. The discrete event simulator ns-3 is a popular network simulator for network research [2].

While different mesh routing protocols like OLSR, AODV, and DSDV have been compared using ns-3 and other network simulators, no such effort has been made with Babel [3], [4]. This is likely due to the absence of an easily accessible Babel implementation for the simulators used.

As far as we know there is no prior implementation of the Babel routing protocol for ns-3. The only other implementation for any simulation environment seems to be an implementation for the event simulator OMNeT++ by Veselý et al. [5].

There exist various other routing protocol implementations for ns-3. Most relevant to this paper is the one for OLSR, as it has served as a reference regarding interactions with ns-3 and the structure of the code in general [6].

2. Background

This section summarizes the aspects of Babel and ns-3 most relevant to this paper.

2.1. Babel

Babel was first published as an Experimental RFC in 2011 [7] and later in January 2021 as a Standards Track RFC [1]. Babel is a robust, proactive, loop-avoiding distance-vector routing protocol, and is suitable for both wired and wireless networks. Being a proactive routing protocol implies it preemptively exchanges routing information for all prefixes, whether there are any packets to be routed or not. A routing loop is a phenomenon where packets get routed in a circle, thereby using up bandwidth while not reaching their destination. In many routing protocols routing loops can form. However, the Babel specification guarantees that no routing loops will ever form if each prefix is originated by only one router. In the case that some prefixes are originated by multiple routers, the specification guarantees that all routing loops quickly disappear and the same loop can never form again [1].

Protocol Operation. In the Babel protocol, routers exchange packets as UDP datagrams sent to a specific neighbor or the multicast address specified in the RFC with a hop count of one. Each packet may contain several messages, called TLVs (Type-Length-Value). There are 11 different types of TLVs in the RFC. However, the protocol is extensible and allows for more types of TLVs to be added. Additionally, most TLVs can contain sub-TLVs.

For neighbor discovery and link quality estimation, each node periodically sends Hello TLVs. Each node computes the receiving cost for each neighbor based on the number of received and missed Hello TLVs from that neighbor. There are two cost computation strategies suggested in the RFC. For wired links, the receiving cost is either a constant value (chosen by the implementation) if the last k out of j Hellos were received, or infinity otherwise. In contrast to wired links, wireless links are not just up or down but have a larger range of link qualities. Therefore the expected transmission count (ETX) metric is suggested: the receiving cost is a constant divided by the fraction of recently correctly received Hellos. This metric has the advantage to favor short, stable links over long, lossy links, which a hop-count-based metric favors [1], [8]. The node regularly sends this receiving cost back to the neighbor inside an IHU TLV (I Heard You). This is necessary since links are not generally symmetric.

Babel can carry prefixes and is, therefore, able to do prefix-based routing. However, its design assumes that each node has a full routing table (to all of the nodes in the network) and is therefore well suited for mesh routing.

The protocol has specific optimizations used on symmetric wired links. For example, it will not resend an update received on a point-to-point link on the same link.

To ensure the strict properties regarding routing loops described above, Babel combines concepts from different routing protocols. The idea Babel uses to (almost) entirely avoid routing loops is the concept of feasibility. Each node maintains for each source (prefix and its originating router) a feasibility distance. This "distance" consists of the newest seqno (sequence number) of the route and the best distance the node has ever announced for this source with the current seqno.

When receiving an update from an Update TLV, a router checks whether the received route has either a newer sequence number or a smaller metric than any it has ever announced. If so, it can be sure not to cause a routing loop by switching to this route. When the topology changes, it might be the case that a router has no feasible routes left. In that case, it sends a seqno request, triggering the source of the route to increment its seqno. After incrementing the seqno, the new route gets forwarded to the router that sent the seqno request [1].

Applications. Babel routers exchange routing information even when there is no mobility event, thus potentially generating unnecessary traffic. Therefore Babel is not the ideal choice for routing in some situations such as large and stable networks and low-power networks. However, Babel is a robust protocol and can be successfully used in most environments. The most prominent use cases include small home networks, heterogeneous networks, and mesh networks [8]

Performance. For the application of mesh networks, multiple experiments conclude that Babels performance is at least comparable - if not better - than specialized mesh routing protocols such as OLSR and BATMAN [9]–[12].

2.2. ns-3

ns-3 is an open-source network simulator first published in 2008 as the successor to the popular network simulator ns2 [13]. It is one of the most widely used network simulators serving as a tool to many network researchers. Like most network simulators, ns-3 is a discrete, event-based simulator: the simulation time is stepped from one event to the next and at each step, all necessary calculations are performed. In terms of both memory usage and computation time, ns-3 is a highly performant simulator capable of large-scale simulations with hundreds or thousands of nodes. It tries to provide a realistic simulation of all network components such as the IP stack or network devices [14].

ns-3 is written in C++ and is structured into modules responsible for different aspects of the simulation. Each with its own tests, examples, and documentation.

3. Implementation

This section outlines the most important aspects of our implementation of the Babel routing protocol for ns-3. Since it is recommended in the Babel RFC to route all control traffic via IPv6, the protocol is implemented as an IPv6 routing protocol.

The implementation is written in C++ and the structure is partially based on the OLSR module included in ns-3 [6]. We provide a new module called `babel` consisting of a simple example network, a helper class to install Babel on existing ns-3 nodes, and the implementation of the protocol itself. The main functionality is located inside the `ns3::babel::RoutingProtocol` class, which extends `ns3::Ipv6RoutingProtocol`. To route IPv6 packets, the methods `RouteOutput` and `RouteInput` are called by the ns-3 IP-stack for outbound and inbound packets respectively. The `ns3::babel::PacketHeader` and `ns3::babel::TLV` classes are responsible for serializing and deserializing Babel packets and the TLVs contained inside them.

ns-3 includes a `TypeId` feature used by the helper class to construct protocol instances. We can add attributes with default values to our `TypeId`, which are used to initialize the objects. This feature is essential since it allows the creator of a simulation to set specific protocol parameters for all nodes or individual nodes. The default values are taken from the RFC. Tunable protocol parameters are, for example, the time intervals used for sending scheduled Hello, IHU, and Update TLVs, as well as the (urgent) timeout for sending messages.

The periodic sending of Hello, IHU, and Update TLVs is governed by Timers set to times specified by the attributes. When a timer expires, we queue the required TLVs for sending on each interface.

To aggregate multiple TLVs into one packet and to apply randomization to the timing of messages, we keep track of a list of queued TLVs and a timer for each interface. Instead of sending a message directly, we instead add it to the queue for later sending and set the timer if it was not already (the timer duration is random within a range specified by the attributes). When the timer expires, it calls a method for sending the packet. This mechanism also allows for the sending of "urgent TLVs" within a shorter timeout. Upon queuing an urgent TLV, the timer is rescheduled to be inside the "urgent timeout" (if it was not already).

The protocol encoding optimizes the size of the packets by not sending redundant information inside each TLV. For example, the Update TLV might not contain the router-id of the router originating this particular route update but relies on a Router-Id TLV preceding it. To follow the encoding, we need to keep track of a parser state for incoming and outgoing packets. Therefore, alongside the queue and timer, we keep track of the parser state of the outgoing packet for each interface. This allows, for example, an Update TLV to add a Router-Id TLV if there was no Router-Id TLV yet or the last Router-Id TLV contained a different router-id.

To receive packets, there is a receiving UDP socket set to listen on the specified multicast address. When a UDP datagram arrives (either as multicast or unicast), the Babel packet in the datagram is delivered by the ns-3 IP-stack to a callback method inside the `ns3::babel::RoutingProtocol` class. Inside this callback method, we deserialize the packet and, while keeping track of the parser state, loop over all TLVs contained inside. If there are any TLVs that require the selected routes to be recomputed, this is done once after all TLVs are processed. The recomputation of the routes may lead to the queuing of new Update TLVs.

The nodes compute their receiving cost using the ETX algorithm outlined in Section 2.1 as the strategy for wireless links.

When routing packets, we need to find the route for the longest matching prefix of the destination address. The route table is a map with the prefix as the key, To allow for fast lookups of routes based on their prefix. Since we do not know the length of the longest prefix and looping over all 128 possible could be costly, we maintain a list of all the prefix lengths we currently store and loop over it instead. A further optimization would be to maintain a tree structure for finding the longest matching prefix or cache route lookups.

Furthermore, we may know of multiple routes for one prefix and, although only one is selected, it is necessary to keep track of the other ones as fallback routes. Maintaining (for each prefix) a list of routes with a pointer to the currently selected one solves this problem while keeping fast access to the selected route.

During the simulation setup in ns-3, when installing the Babel routing protocol on a node using the Babel helper class, it is possible to exclude interfaces from the Babel protocol. This way, a network of Babel nodes can link to the other nodes, possibly using another routing protocol. All Babel nodes originate all of their global addresses as well as the prefixes of the excluded interfaces. To illustrate, consider the network in Figure 1, where `R` and `A` are Babel nodes and `S` is another server.

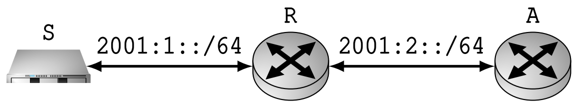


Figure 1: Network Topology

During the setup of `R`, its left interface (to `S`) has been excluded from Babel routing. Therefore, `R` originates the prefix of this network (`2001:1::/64`) along with its global addresses.

3.1. Capabilities

Our implementation is a working IPv6 routing protocol for ns-3. Considering the only other routing protocol for IPv6 is currently `ns3::Ipv6StaticRouting`, just having a non-static routing protocol might already be helpful in some [15] scenarios.

The main point of the implementation, and hence its most relevant capability, is to accurately depict the behavior of Babel inside ns-3. This is achieved by following the specification and further demonstrated in Section 4.

The use of ns-3 attributes makes it easy to tune protocol parameters to meet the needs of a specific simulation. A performance comparison with different protocol parameters is also possible.

Since ns-3 has the option to trace all packets to a file and we serialize all packets as described by the specification, it is possible to use a tool such as Wireshark to inspect Babel packets exchanged during a simulation.

3.2. Limitations

The goal of the current implementation is to provide a working version of Babel for simulations of the protocol behavior in different environments. As of now, it is lacking some features and does not yet comply with all aspects of the RFC. Most aspects of non-compliance are not an issue since they are not strictly required for the protocol, and our protocol instances only communicate with other nodes inside ns-3 using the identical implementation. In other words, for simulations inside ns-3, no interoperability with other protocol implementations is needed.

Most notably, the routing protocol is currently an IPv6 routing protocol and only supports IPv6 traffic. The recommended way to use Babel is to have a single protocol instance that routes IPv4 and IPv6 traffic but communicates exclusively using IPv6 [1]. Such behavior can most likely be achieved in ns-3. However, since most simulations use either IPv4 or IPv6 it would be desirable to have a standalone IPv4 implementation as well.

There is currently no recognition of Sub-TLVs, Unicast Hellos, Acknowledgments, Acknowledgment Requests and some encoding methods as defined in the RFC. As mentioned before, if this implementation does not need to interoperate with others this is not a problem.

4. Tests

To demonstrate the functionality of the implementation, we devised a test scenario. The setup consists of 6 nodes connected on six point-to-point links as shown in Figure 2. Nodes `R`, `A`, `B`, `C`, and `D` are Babel routers. After allowing the protocol a brief initialization time, nodes `A`, `B`, `C`, and `D` start emitting 20 UDP packets per second to `S`. At 35 seconds into the simulation they stop sending the packets. At 20 seconds, the link between `R` and `A` gets cut. To route packets to `S`, the routers use their routes to `2001:6::/64`, a prefix originated by `R`.

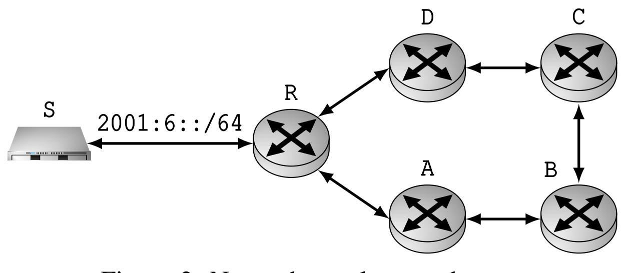


Figure 2: Network topology under test

Node `S` tracks the number of packets arriving, and its results are shown in Figure 3. While this graph illustrates the routes leading to `S`, the protocol also tracks all other routes, which are not visualized here. For clarity, the packets from `C` and `D` are not shown since they are not affected by the link being cut and all packets arrive as expected.

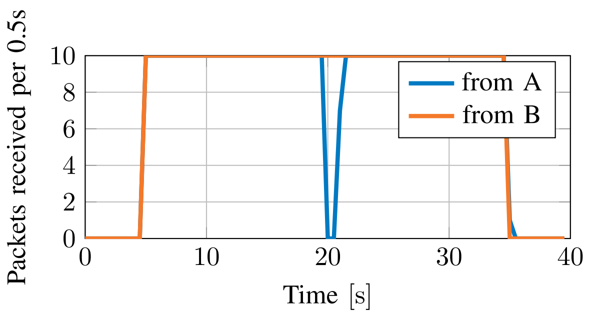


Figure 3: Packets Received at S

Just before cutting the link, node `B` has two routes to `2001:6::/64` (and therefore `S`). One route via `C` and another via `A`. When `A` detects the link is cut, it sends a route retraction for all routes it would forward to `R`. When receiving this retraction, `B` switches to its route via `C` and sends an update. A can, however, not switch to this route via `C`, as it is not feasible. Essentially, `A` has no way of knowing that adopting this route would not create a routing loop. Left with no routes, `A` sends a seqno request, which gets forwarded by `B`, `C`, and `D`, until it reaches `R`. `R` increments its seqno and sends an update with the new seqno, which is quickly forwarded back to `A`. After receiving an update with the new seqno, `A` can now switch to a route via `B`. During the time it takes the seqno request and updates to go around the network, `A` has no route to `2001:6::/64` (and therefore `S`). This can be seen in Figure 3 as the drop in packets received from `A`. The increased number of packets traveling to `S` via `D` is still below the capacity of the links, so there is no drop in the packets received from the other nodes.

And the end, the route table of `A` for routes to `2001:6::/64` looks as follows (advertised metric is the metric announced by the neighbor. The cost of a route is the advertised metrix plus the cost of the link to the neighbor):

- next hop: `fe80::200:ff:fe00:4`;
  advertised metric: 768; seqno: 0x8001
- next hop: `fe80::200:ff:fe00:1`;
  advertised metric: 0; seqno: 0x8000

`fe80::200:ff:fe00:4` is the link-local address of an interface of B. This is the selected route with a seqno one higher than the other route. `fe80::200:ff:fe00:1` is the link-local address of an interface of R. Although the advertised metric is 0, the metric overall is infinity since the link cost from A to R is infinity.

5. Conclusion and Future Work

This paper introduced an implementation of the Babel routing protocol for the discrete event simulator ns-3. This implementation can be used to help research applications and the performance of Babel using ns-3.

Although the current state of our work suffices to simulate the operation of Babel, it is desirable to finish the implementation to comply with the RFC (see Section 3.2).

To fulfill the goal of providing an easy way to simulate the behavior of the Babel protocol, the behavior inside the simulation must match the behavior in the real world. Therefore, it is vital to validate the results from the simulation with results obtained in the real world. This requires either carrying out hardware tests or recreating an existing test setup inside the simulator.

An interesting idea, which would be easy to test now, is writing an extension to optimize protocol performance in fast-moving mobile ad-hoc networks by relaying position information through the protocol. A similar idea using a custom OLSR implementation shows promising results in [16], and it is interesting to see how that compares to a Babel version.

References

[1] J. Chroboczek and D. Schinazi, "The babel routing protocol," Internet Requests for Comments, RFC Editor, RFC 8966, January 2021.

[2] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[3] D. Bhatia and D. P. Sharma, "A comparative analysis of proactive, reactive and hybrid routing protocols over open source network simulator in mobile ad hoc network," *International Journal of Applied Engineering Research*, vol. 11, no. 6, pp. 3885–3896, 2016.

[4] R. K. Jha and P. Kharga, "A comparative performance analysis of routing protocols in manet using ns3 simulator," *International Journal of Computer Network and Information Security*, vol. 7, no. 4, pp. 62–68, 2015.

[5] V. Veselý, V. Rek, and O. Ryšavý, "Babel routing protocol for omnet++ - more than just a new simulation module for inet framework," 2016.

[6] "Optimized Link State Routing (OLSR) — Model Library - NS-3," https://www.nsnam.org/docs/models/html/olsr.html, accessed: 2021-06-12.

[7] J. Chroboczek, "The babel routing protocol," Internet Requests for Comments, RFC Editor, RFC 6126, April 2011.

[8] ——, "Applicability of the babel routing protocol," Internet Requests for Comments, RFC Editor, RFC 8965, January 2021.

[9] D. Murray, M. Dixon, and T. Koziniec, "An experimental comparison of routing protocols in multi hop ad hoc networks," in *2010 Australasian Telecommunication Networks and Applications Conference*, 2010, pp. 159–164.

[10] M. E. Villapol, D. Pérez Abreu, C. Balderama, and M. Colombo, "Comparación del rendimiento de los protocolos de enrutamiento para redes malladas en una red experimental con restricciones de ancho de banda en el enrutador del borde," *Revista de la Facultad de Ingeniería Universidad Central de Venezuela*, vol. 28, no. 1, pp. 7–13, 2013.

[11] M. Abolhasan, B. Hagelstein, and J.-P. Wang, "Real-world performance of current proactive multi-hop mesh protocols," in *2009 15th Asia-Pacific Conference on Communications*. IEEE, 2009, pp. 44–47.

[12] J. Pramod, K. Sahana, A. Akshay, and V. Talasila, "Characterization of wireless mesh network performance in an experimental test bed," in *2015 IEEE International Advance Computing Conference (IACC)*. IEEE, 2015, pp. 910–914.

[13] "ns-2 and ns-3," https://www.nsnam.org/support/faq/ns2-ns3/, accessed: 2021-06-10.

[14] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–5.

[15] "Ipv6 - model library - ns-3," https://www.nsnam.org/docs/models/html/ipv6.html, accessed: 2021-06-10.

[16] S. Sharma, "P-OLSR: Position-based optimized link state routing for mobile ad hoc networks," in *2009 IEEE 34th Conference on Local Computer Networks*. IEEE, 2009, pp. 237–240.

ISBN 978-3-937201-73-3
DOI 10.2313/NET-2022-01-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)