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 2019

Munich, Germany

Editors

Georg Carle, Stephan Günther, Benedikt Jaeger

Publisher

Chair of Network Architectures and Services

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

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

Summer Semester 2019

Munich, February 22, 2019 – September 1, 2019

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET 2019-10-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Summer Semester 2019

Editors:

Georg Carle
Chair of Network Architectures and Services (I8)
Technical University of Munich
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 19
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, February 22, 2019 – September 1, 2019
ISBN: 978-3-937201-68-9

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2019-10-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2019-10-1
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 2019. 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 Dominik Spörle with the paper *Surveying the depth of user behavior profiling in mobile networks* and Marco Weiss with the paper *Optimization of Decision Trees for TCP Performance Root Cause Analysis*.

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, October 2019

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

Simon Bauer (bauersi@net.in.tum.de)
Technical University of Munich

Paul Emmerich (emmericp@net.in.tum.de)
Technical University of Munich

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

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

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

Jonas Jelten (jelten@net.in.tum.de)
Technical University of Munich

Marton Kajo (kajo@net.in.tum.de)
Technical University of Munich

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

Maurice Leclaire (leclaire@in.tum.de)
Technical University of Munich

Stefan Liebald (liebald@net.in.tum.de)
Technical University of Munich

Christian Lübben (luebben@net.in.tum.de)
Technical University of Munich

Johannes Naab (naab@net.in.tum.de)
Technical University of Munich

Dominik Scholz (scholz@net.in.tum.de)
Technical University of Munich

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

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

Seminar Homepage

`https://net.in.tum.de/teaching/ss19/seminars/`

Contents

Block Seminar

Seminar

DDS vs. MQTT vs. VSL for IoT

Georg Aures, Christian Lübben*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: g.aures@tum.de, luebben@net.in.tum.de

***Abstract*—Connecting IoT devices is a task, developers have to solve, when they would rather concentrate on application and hardware. This survey contributes an overview over the three different middlewares Data Distribution Service (DDS), Message Queuing Telemetry Transport (MQTT), and Virtual State Layer (VSL) which are compared from a developers point of view. Evaluation focuses on how easy the protocols can be used and on how much work is taken away from the developer to be automated in the middleware for regular tasks like securing, searching, and serializing the data. Transparent of their actual implementation the compared key features are presented in a rating table to provide the architect of IoT infrastructure with a guide on which protocol is suitable for which use case.**

***Index Terms*—software-defined networks, feature comparison, rating measures, Internet of Things, middleware**

1. Introduction

Data exchange is a fundamental part of distributed systems. Different IoT middleware use different techniques to address the key features *security*, *data modeling* and *practical usability*. This paper compares the protocols Data Distribution Service (DDS), Message Queuing Telemetry Transport (MQTT), and Virtual State Layer (VSL) with measures in those three key features. This overview should enable the developer of an IoT application to carefully select the appropriate middleware.

2. Background

This section gives an short description of the compared network middleware technologies. The compared middlewares differ in their network topology and their features.

2.1. Data Distribution Service

DDS is a data-centric publish-subscribe middleware for highly dynamic distributed systems [1], which is standardized by the Object Management Group [2]. The network topology is discovered dynamically and connections between nodes are established peer-to-peer without a central server as a single point of failure. There is a rich set of Quality of Service (QoS) policies which are a reason why DDS is typically used in industrial environments [3].

2.2. Message Queuing Telemetry Transport

MQTT is an open message protocol standardized by the Organization for the Advancement of Structured Information Standards (OASIS) [4]. Clients can publish and subscribe data under topics (like *kitchen/oven/temperature*) and a central server is forwarding the messages to the subscribers. The main focus is on small code footprint and low network bandwidth [1].

2.3. Virtual State Layer

The Virtual State Layer (VSL) is a site-local, data-centric architecture for the IoT. Special features are full separation of logic and data in IoT services, explicit data modeling, a semantical data lookup, stream connections between services, and security-by-design [5].

3. Related work

In [1] Profanter et al. compare the performance of DDS and MQTT (and other protocols). In [6] Sarafov compares the overhead of the WebSocket, CoAP and MQTT protocols. Thota [7] compares the lightweight protocols MQTT and CoAP. Those papers have in common that they implement the protocols in a testing environment to evaluate some of the protocol features in detail.

This survey paper collects the work from several evaluation, implementation and architectural papers in order to give an *overview* over the compared protocol features and their differences. This survey is restricted to a selection of key features (restriction in breadth) which are evaluated from a protocol users point of view – leaving out implementation details and underlying technology (restriction in depth).

4. Survey Approach

This survey provides an overview over some extracted measurements, based on the related work and other research on performance and evaluation of DDS, MQTT and VSL for IoT. The evaluation of new measurements is not in the reach of this paper. The selected measures are presented in a table and a short description for each rating is given in section 5 comparison. The measures can be divided into three feature categories.

Security In many setups IoT devices are connected via an insecure connection over the Internet. Security is a crucial feature for many IoT applications e.g. personal,

doi: 10.2313/NET-2019-10-1_01

medical or critical industrial. Here the measures *data integrity*, *authentication*, *access control*, and *encryption* are rated because for some applications they can be even mandatory by law [8].

Data modeling for convenient discovery and access at application level and the means of transport and storage in a way that they are ideally transparent to the user are covered by the measures *data gnostic*, *data centric*, *serialization*, *protocol overhead*, and *QoS*.

Practical usability is covered by the measures *simplicity of use*, *real world testing*, and *monitoring & RTM* because they have a strong influence on the resources, which have to be spend for application development in IoT.

5. Comparison

The compared network technologies for IoT and a short description of their abilities which are rated in table 1 with four grades: feature not available (–), basic coverage of some aspects (+), feature is fully implemented (++), with additional benefits (+++).

5.1. Data integrity

The integrity of data means that it is unaltered and consistent [9].

DDS. The built-in authentication plug-in uses public key infrastructure (PKI) with a trusted identity certificate authority. Each DDS domain participant is certified by the certificate authority (++) [10, min. 9 f.].

MQTT. Performing integrity checks is left to the application [4]. Even though not part of the protocol specification, some implementations like HiveMQ[1] support integrity checks. Still if a features is not specified in the protocol it might not be compatible between implementations (–). E.g. in the case of integrity checks, different implementations can use different hash functions.

VSL. The integrity of data and executables is checked with a certificate [9]. Also the integrity of other files (e.g. metadata files) can be protected with a cryptographic hash certificate (+++) [9].

5.2. Authentication

To verify the identity of a peer it has to authenticate itself [9].

DDS. Authentication of every entity that produces or consumes data in the network (++) via public key infrastructure [1] [11, min. 38] [12, p. 53-64].

MQTT. Basic authentication mechanism based on usernames and passwords is supported [13]. These credentials are sent with the CONNECT message, further authorization is not provided [14]. Only good for secure channels (+), since password is sent in clear text [4].

1. https://www.hivemq.com/blog/mqtt-security-fundamentals-mqtt-message-data-integrity/

VSL. Not only participating entities are authenticated but also the preceding entities in the processing chain [9]. Nodes can authenticate others locally with cached certificates which are autonomously renewed by a certificate management [9]. Automatizing the important renewal of certificates provides additional security (+++)

5.3. Access control

Access control to services is a key security feature and mandatory for certain environments – for example for infrastructure in Germany [8].

DDS. The common access control settings are configured in a governance file for the hole domain [10, min. 10]. Permission documents signed by a certificate authority describe what each participant is allowed to do within the domain. The PermissionsHandles can cache any QoS that is relevant to access control decisions Access Control Plugin [12, p. 65-71]. Full access control is implemented, which can be tedious to apply separately for each participant (++).

MQTT. The protocol itself does not specify access control (–) [15]. Some implementations like mosquitto[2] or HiveMQ[3] implement access control via user/password or RSA authentication for the subscriber and publishers for specified topics.

VSL. Role-based access control is implemented [16], [17]. This provides additional security because roles can be used in a way that only the necessary access is granted (+++). Not only the type information but also the access modifiers are synchronized over the network, to filter the discovery results based on a service's access ID already at the source [5]. Most important, it takes the burden to implement adequate service access security from the developers [5].

5.4. Encryption

The goal of encryption is to protect the data from unauthorized readers.

DDS. The build in Cryptographic Plugin uses AES in counter mode [18]. The plugin can configured to only encrypt some topics (++). [12, p. 72-84].

MQTT. Because encryption is not supported (–) by the protocol [13] some implementations like HiveMQ [19] suggest encryption mechanisms on the application layer. Generally encryption on transport layer is recommended for MQTT [13].

VSL. All communication between peers is encrypted (++) [20].

2. https://mosquitto.org/man/mosquitto-conf-5.html
3. https://www.hivemq.com/docs/4/control-center/configuration.html#access-control

doi: 10.2313/NET-2019-10-1_01

5.5. Data gnostic

Knowledge of the data structure and content enables the use of simple logic like aggregation and plausibility checks close to the data. In large scale systems keeping track of the meta information becomes too complex to be handled by separate development documentation.

MQTT. A client can publish and subscribe data under a topic. The topics are hierarchically structured and can also be accessed vertically [4]. The data itself is not known to the protocol, which makes it necessary for the applications to agree on the structure of data on a meta-level for example a possibly outdated documentation file (+).

DDS. (++) Each topic is bound to a data-type [3]. The data-type and the labels describe the data in a machine readable way, so the protocol is data *gnostic* (Greek gnostos “known, perceived, understood”).

VSL. More descriptive than typed topics, VSL is structured with searchable (+++) hierarchical context models, which are stored in a repository [5]. In virtual nodes the data in the VSL can be dynamically overlayed by a service which provides live data only when requested [21] [22].

5.6. Data centric

Data centric protocols provide an abstraction for the messages send between peers and automate keeping track of shared variable states. In contrast to message centricity, data centricity decreases implementation complexity and time.

DDS. Data is published into the DDS domain and subscribers can subscribe without prior knowledge where the information comes from or how it is structured, as the package already describes itself [2]. Dynamic discovery of topics without a central instance, self describing data packages and transparent data sources [2] make to protocol truly data centric (++).

MQTT. As a “Message Queuing” protocol, the application needs to keep track of the variable states itself (–). For IoT this is a serious issue, because the number of devices and their states, for which each application has to keep track can be very large.

VSL. “Through its separation of service logic and data it offers more functionality by-design such as security [than DDS].” [5] VSL is even “information centric”, because it provides full data management including data modeling, discovery, caching, and security, which is an advantage over pure data centricity (+++).

5.7. Serialization

Compressing the data for transmission over the network is a task that should be taken care of the network technology and not by the application, because decoding data is error prone if an explicit coupling with meta data is missing.

TABLE 1: Overview of the compared features

measure	DDS	MQTT	VSL
data integrity	++	−	+++
authentication	++	+	+++
access control	++	−	+++
encryption	++	−	++
data gnostic	++	−	+++
data centric	++	−	+++
serialization	++	−	++
protocol overhead	−	+++	−
QoS	+++	+	+
simplicity of use	−	++	++
real world testing	++	+	++
monitoring & RTM	+++	+	++

DDS. The data is serialized for network transmission without any further information needed (++), since the topics are typed (eg. "float temperature") [3].

MQTT. The protocol does not support serialization of the data, which has to be un/marschalled by the application.

VSL. In [22, section 4.2] Kuperjans describes the serialization of the VSL data structures in XML, JSON, CBOR and Google protocol buffers. No additional information is required, because the data is completely described in the context model repository (++).

5.8. Protocol overhead

The protocol overhead has an negative impact on the network performance which is especially of interest when the network capacity is low.

DDS. Because different kinds of data can be sent in a single package, the payload needs additional identification data [1]. Also diagnostics information can be send with every transmitted data package [1]. The discovery phase for the network and periodically heartbeat packages can add additional overhead which depends not at least on the chosen implementation of the DDS protocol [1]. All this meta information generates overhead to the core data (–).

MQTT. A dedicated TCP connection is created for every subscriber and publisher pair, therefore it is unnecessary to include additional information about the published data in the transmitted package. In [1] Profanter et al. show that MQTT not only adds the smallest amount of additional data during the connection initialization, it also has a very small overhead when sending out data messages, compared to other protocols (+++).

VSL. There is a notable overhead (–) which is caused by so-called alive pings as well as the self-management properties of the network [21]. In [9] Pahl and Donini describe a mechanism to disperse to overhead resulting from certification.

5.9. Quality of Service

Quality of Service (QoS) describes the ability to configure performance and reliability of the network.

doi: 10.2313/NET-2019-10-1_01

DDS. The data in a topic is associated with a specific configuration from a broad set of QoS parameters e.g. durability, lifespan, presentation, reliability, and deadlines [2]. Detailed QoS is the strength of DDS (+++) with a separate QoS contract between every data reader and writer [3].

MQTT. QoS is defined in three levels, so that messages are send: at most once, at least once or exactly once [4]. It can be specified if the server should cache data [1]. This basic QoS parameter leaves a lot of other configurations (like the update frequency) to application level (+).

VSL. In [23] Pahl and Liebald describe a *Modular Distributed IoT Service Discovery*, where one of the goals is discovering the service provider with the best latency. Using this mechanism the VSL serves each client with the best discovered latency. In [24] Pahl, Liebald and Lübben demonstrate how VSL performs running on top of existing internet technologies at the example of a complex application which can still provide a real-time user experience. Together with the virtual nodes (see section 5.5, [21]) the concept of always providing best available quality provides basic coverage of this feature (+).

5.10. Simplicity of use

Comparison on how fast new users can learn the protocol and how convenient tasks of various complexity can be solved.

DDS. Lars Mijeteig states in a youtube video [25, min. 17] that DDS can be hard to start grasping because of its many options (–).

MQTT. A lightweight, simple protocol, which makes it simple to use (++) [26]. Still it should be kept in mind, that features like encryption have to be taken care of by the application [19]. In this case the simplicity of MQTT would introduce complexity elsewhere.

VSL. For evaluation of usability the authors of [5] let 150 IoT-beginners implement a complex use case after solving a tutorial. 73% rated the VSL API as "well suitable or even easy-to-use for beginners" and all managed to complete their project in less than 20h [5]. With its high degree of automation VSL is both simple and powerful (++).

5.11. Real world testing

The possibility to implement testing functionality and run it in a integration or production environment.

DDS. According to Mijeteig [25, min. 8] test functionality can be added in a plug and play manner (++).

MQTT. Under [27] several tools are listed that support MQTT real world testing. This shows that real world testing is possible (+), but still not part of the protocol.

VSL. In [28] Pahl states that continuous "real world testing" is a requirement, because each IoT site is different, making comprehensive service testing before deployment difficult. With a sophisticated application one could imagine even automated testing (++).

5.12. Monitoring and runtime management

The ability to manage the network at runtime (RTM) is crucial for situations where downtime is very costly. Monitoring is also a key tool to ensure high uptime rates.

DDS. (+++) There is a dedicated topic to log security-relevant messages [12, p. 87-88]. Mechanisms to monitor presence, health and activity of all entities are available and a concept of liveliness is supported. With a concept of deadline DDS can monitor the activity of each individual data-instance in the system. If an instance is not updated according to the requirements of the receiving application, the application is notified. With a concept of lifespan DDS understands if a data-object has outlived its purpose and is considered 'stale' data. [29, 39-40]

MQTT. Tools for monitoring have to be used separately[4]. Due to the simplicity of the protocol, primitive runtime management is possible by introducing new subscribers/publishers to the broker (+).

VSL. (++) Short lifetime certificates enable service meta data changes at runtime [17]. Models can also be created at runtime, where they only affect the local model repository [20].

6. Evaluation

The comparison shows that the protocols each have unique strengths, so that no protocol is dominated by another with an always better or equal rating through all compared features. MQTT has a very small protocol overhead and is simple to use. DDS with its rich Quality of Service properties, monitoring and runtime management is a good choice in an industrial setting where experienced developers have access to all the applications in the network. VSL is easy to use and has a rich data discovery mechanism with build in security which makes it suitable also for inexperienced developers who want to integrate applications into distributed ecosystem of applications which are not known to the developer.

7. Conclusion (and future work)

Given the results of the comparison there is a good reason to choose each for the three protocols given a specific use case. Table 1 shows an overview from a developers point of view with the corresponding reasons for the rating discussed and cited to detailed papers in section 5.

Further work could evaluate features like *simplicity of use* or *real world testing* in direct comparison of the three protocols, for example with testers who have to implement the same task with all three network technologies. Also an evaluation on how the data is structured semantically and possibilities of increasing data availability with means of the middleware are interesting for further comparison.

4. http://www.steves-internet-guide.com/mqtt-tools/

doi: 10.2313/NET-2019-10-1_01

References

[1] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019. [Online]. Available: http://mediatum.ub.tum.de/doc/1470362/1470362.pdf

[2] "About the data distribution service specification version 1.4," OMG website, Mar. 2015. [Online]. Available: https://www.omg.org/spec/DDS/1.4/

[3] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206. [Online]. Available: https://ieeexplore.ieee.org/document/1203555

[4] A. Banks and R. Gupta, "MQTT version 3.1. 1 plus errata 01," *OASIS standard*, vol. 29, p. 89, Dec. 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[5] M.-O. Pahl and S. Liebald, "Information-centric IoT middleware overlay: VSL," in *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Mar. 2019. [Online]. Available: https://s2labs.org/download/publications/2019-03_NetSys_Designing_a_Data-Centric_Internet_of_Things.pdf

[6] V. Sarafov, "Comparison of IoT data protocol overhead," *Network Architectures and Services, Website: https://www. net. in. tum. de, accessed on*, vol. 23, 2018. [Online]. Available: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2018-03-1/NET-2018-03-1_02.pdf

[7] P. Thota and Y. Kim, "Implementation and comparison of m2m protocols for internet of things," in *Proc. 4th Int. Conf. Appl. Comput. Inf. Technol.* IEEE, 2016, pp. 43–48. [Online]. Available: https://ieeexplore.ieee.org/document/7916956

[8] "BSI act of 14 August 2009 (federal law gazette I p. 2821) last amended by article 1 of the act of 23 June 2017 (federal law gazette I p. 1885)," passed by the Bundestag as Art. 1 of the Act of 14/8/2009 I 2821, Aug. 2009. [Online]. Available: https://www.bsi.bund.de/EN/TheBSI/BSIAct/bsiact_node.html

[9] M.-O. Pahl and L. Donini, "Securing IoT microservices with certificates," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8406189

[10] A. Mitz, "Revolutionizing data distribution with an open and secure DDS," OCI webinar, Aug. 2018. [Online]. Available: https://youtu.be/yDY3lOf4XhU

[11] G. Pardo-Castellote, "Data Distribution Service™ (DDS™)," Object Management Group talk, Dec. 2018. [Online]. Available: https://youtu.be/6iICap5G7rw

[12] ——, "OMG data-distribution service security," LinkedIn Corporation, Mar. 2014. [Online]. Available: https://www.slideshare.net/GerardoPardo/omg-datadistribution-service-security

[13] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, "Authorization mechanism for MQTT-based internet of things," in *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2016, pp. 290–295. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7503802

[14] "Authenticating & authorizing devices using MQTT with auth0," Auth0® documentation. [Online]. Available: https://auth0.com/docs/integrations/authenticating-devices-using-mqtt

[15] Y. Upadhyay, A. Borole, and D. Dileepan, "MQTT based secured home automation system," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. IEEE, 2016, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7570945

[16] R. S. Sandhu, "Role-based access control," in *Advances in computers*. Elsevier, 1998, vol. 46, pp. 237–286. [Online]. Available: https://doi.org/10.1016/S0065-2458(08)60206-5

[17] M.-O. Pahl and L. Donini, "Giving IoT services an identity and changeable attributes," in *International Symposium on Integrated Network Management (IM), Washington DC, USA*, Apr. 2019. [Online]. Available: https://s2labs.org/download/publications/2019-04_IM_Giving_IoT_Services_an_Identitiy_and_Changeable_Attributes.pdf

[18] R. Housley, "Using advanced encryption standard (AES) counter mode with ipsec encapsulating security payload (ESP)," The Internet Society, Jan. 2004. [Online]. Available: https://tools.ietf.org/pdf/rfc3686.pdf

[19] "MQTT security fundamentals: MQTT payload encryption," 2015. [Online]. Available: https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption/

[20] M.-O. Pahl and G. Carle, "The missing layer — virtualizing smart spaces," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE, 2013, pp. 139–144. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6529471

[21] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed smart space orchestration," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 979–984. [Online]. Available: https://www.pahl.de/download/publications/NOMS2016_Distributed_Smart_Space_Orchestration_Pahl.pdf

[22] F. Kuperjans, "Native service interfaces for the virtual state layer," Master's thesis, Technische Universität München Department of Informatics, 2017. [Online]. Available: https://www.net.in.tum.de/fileadmin/bibtex/publications/theses/ba-kuperjans.pdf

[23] M.-O. Pahl and S. Liebald, "A modular distributed IoT service discovery," in *International Symposium on Integrated Network Management (IM), Washington DC, USA*, 2019. [Online]. Available: https://s2labs.org/download/publications/2019-04_IM_A_Modular_Distributed_IoT_Service_Discovery.pdf

[24] M.-O. Pahl, S. Liebald, and C. Lübben, "VSL: A data-centric internet of things overlay," in *2019 International Conference on Networked Systems (NetSys'19)*, Mar. 2019. [Online]. Available: https://s2labs.org/download/publications/2019-03_NetSys_Demo_VSL.pdf

[25] L. I. Miljeteig, "Data Distribution Service - Lars Ivar Miljeteig," NDC Conferences, Dec. 2017. [Online]. Available: https://youtu.be/3p-iVgWItJ8

[26] "MQTT 101 - how to get started with the lightweight iot protocol," 2014. [Online]. Available: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/

[27] J. Colantonio, "Top IoT testing tools that support MQTT," Automation Testing, Mar. 2019. [Online]. Available: https://www.joecolantonio.com/iot-testing-tools/

[28] M.-O. Pahl, "Multi-tenant IoT service management towards an iot app economy," in *HotNSM workshop at the International Symposium on Integrated Network Management (IM), Washington DC*, 2019. [Online]. Available: https://s2labs.org/download/publications/2019-04_IM_HotNSM_Multi-Tenant_IoT_Service_Management_towards_an_App_Economy.pdf

[29] G. Pardo-Castellote, "Introduction to OMG DDS," LinkedIn Corporation, Mar. 2013. [Online]. Available: https://www.slideshare.net/GerardoPardo/introduction-to-omg-dds-1-hour-45-slides

The Basics of Multi Signatures using RSA

Jonas B. Erasmus, Dr. Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: jonas.erasmus@tum.de, kinkelin@net.in.tum.de

***Abstract*—Requiring multiple parties to generate a single signature can improve security and accountability. This paper presents a scheme for Certificate Authorities to have two signers generate and verify a single certificate, without the client noticing. Additionally shortcomings and alternatives to multi key signatures are explored.**

***Index Terms*—Digital signature, Multi-Signature, RSA, Public Key Cryptography, Certificate Authority, Secret Sharing**

1. Introduction

In the internet almost all traffic is authenticated, the sender signs the message to prove it was sent by them. While this system works efficiently on a personal basis, larger entities or companies also sign messages using only one key. As a result, whoever knows that key may authenticate messages. While this is a security risk, additionally the question arises if a single person should be able to sign in the name of the whole company. Certificate Authorities for example sign the certificates of websites. Even with the whole verification process for a domain running successfully, in the end a single key (and as such a single person) signs the certificate. The need to verify messages is without question. But with a single signing key, whoever is in control of this key can officially speak for the whole company. While this is an issue of trust, there also is the risk of human error. And even if a misuse is noticed, the tracing of who used the signing key for what grows ever more complicated with the size of the company. This paper presents a solution of multiple keys being used to generate a single signature. This can enforce multiple signers checking the signature and only if every participant is satisfied a valid signature is generated. Additionally none of the signers is in possession of the master key, but only knows a key part.
First the basic concept of signing is explained using RSA. Then Colin Boyd's suggestion of an dual signer approach [1] is presented, and applied to certificate issuing. Finally the shortcomings and alternatives to multi key signatures are explored and evaluated.

2. Fundamentals

For the scope of this paper, only the signing of messages using RSA (Rivest–Shamir–Adleman) will be investigated. As such, the main focus is on the keys used by the signing entity. For basic operation only two keys are required. Key *e* is the public key to be shared with all other users, while *d* represents the private key that is kept hidden from other entities. For the sake of simplicity key generation and distribution is omitted.

2.1. RSA

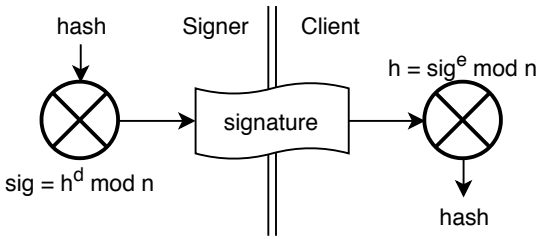


Figure 1: Basic RSA signing scheme

The RSA cipher (as depicted in Fig.1) is widely used in today's internet to sign and verify the authenticity of messages. The to be signed message *M* is hashed to a hash-value *h*.

$$h_s = hash(M) \tag{1}$$

Subsection 5.2 of the corresponding RFC [2] (Request-for-Comments; Part of the Internet standardization) describes the algorithm as follows. With the given *h* as the hashed message, and the key *(d, n)* (private key, and publicly known modulus) a signature *sig* may be computed.

$$sig = h_s^d \mod n \tag{2}$$

For further use we will refer to signing a Message *M* with Key *d* using RSA as *RSA(M, d)*. The signature is then appended to the message and the finished package is sent to the receiver. Here the message can be verified using the known public key *e* of the signer.

$$h_r = sig^e \mod n \tag{3}$$

$$h_r \stackrel{?}{=} hash(M) \tag{4}$$

Should (4) hold true, the receiver knows, that the message was indeed sent by the signer. As an imposter could not generate the correct signature for their message (since they cannot know the required private key).
In a minimal notation RSA signing can be written as

$$sig = RSA(M, d) \tag{5}$$

for signing, and

$$hash(M) \stackrel{?}{=} RSA(sig, e) \tag{6}$$

for verification of a signature.

doi: 10.2313/NET-2019-10-1_02

2.2. Important RSA Properties

The usage of multiple keys in the RSA algorithm is straight forward. A signature may be signed a second time, however the result is the same as if signing with a "concatenated" key (explained in depth in Sec.3.2).

$$RSA(RSA(M, d1), d2) = RSA(M, d1.d2) \quad (7)$$

Additionally RSA is commutative, the key order for multi-signing is irrelevant.

$$RSA(RSA(M, d1), d2) = RSA(RSA(M, d2), d1) \quad (8)$$

By using this property for two random keys *d1* and *d2*, a third public key *e* can be found fulfilling the public/private key properties

$$hash(M) = RSA(RSA(M, d1.d2), e) \quad (9)$$

As such *d1* and *d2* can be interpreted as two parts of a private key. When choosing n ($n > 1; n \in \mathbb{N}$) random keys, an inverse key (that is a key to decrypt the message) can always be found.
The security of the use of multiple RSA steps is still enforced. Without knowing the master key *d1.d2*, it is not possible to derive *d1* when only knowing *d2* and *e* (or vice versa). The splitting is only defeated once multiple key parts of the private key are known to the same entity. It could then compute the concatenation of those keys, obtaining a larger key part of the master key (or the master key itself if knowing all key parts).

3. Multi Key RSA Signing

This section describes how the RSA algorithm can be extended to allow multiple parties to generate a single signature.

3.1. Goal

The important part of multi key signing is that a client should be unable to tell a difference to a "single-signed" signature. As such the signer may use a multi key scheme, however there is still only one public key and the receiver (client) can verify the signature using the common RSA verification. In order to do so the private key is split into multiple keys, however the public key only verifies the concatenation of all private key parts.
Additionally the signers may not be able to generate a valid signature on their own. Multiple signers need to sign in order for the signature to be correct. Still a central authority, rolling out the individual key parts, is needed.

3.2. Key Distribution

In order to construct all the partial keys, full information of all involved keys is needed. To derive the partial keys the *master key* (the 'un-spilt' private key) needs to be known to compute the corresponding *public key*. As such a central authority needs to compute all the partial keys *d1* to *dn* and the public key *e*.
By using the standard key generation algorithm so called 'backdoor' information is generated. Using this extra knowledge an inverse key can be computed. It is also possible to find the inverse of multiple concatenated keys (by treating the product over all private keys as the master key). While computation time is higher than normal (the possibly large keys need to be multiplied) key generation only needs to be done once.
The partial keys then need to be distributed (via a secure channel) to all signers participating in the process.
The following explanations assume the partial keys were successfully distributed and the public key *e* is well known by all entities. The modulus *n* is a constant (all keys and partial keys are of the same "length") and as such is known, or even hard-coded into the RSA algorithm.

3.3. The Basic Idea

This subsection is a recollection of Boyd's suggestion [1] for two signers as depicted in Fig.2.

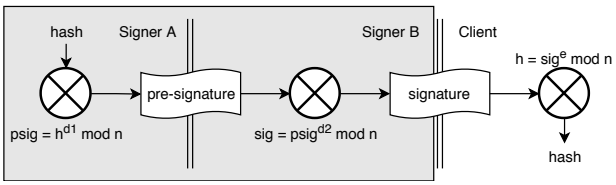


Figure 2: Signing using two Signers

With two signing parties, three keys are needed. The private key is split into two parts *d1* and *d2*, and the public key *e*.
The first signer signs the message using its own (secret) key part *d1* generating an incomplete signature.

$$psig = RSA(M, d1) \quad (10)$$

The second signer can now create the final signature using their own private key (11). This signature should not be published, unless (12) holds true (the second signer checks the correctness of $psig$).

$$sig = RSA(psig, d2) \quad (11)$$

$$hash(M) \stackrel{?}{=} RSA(sig, e) = RSA(psig, d2.e) \quad (12)$$

For a client this process is transparent, the public key *e* is the only information needed to verify the final signature. Since RSA is commutative (Sec.2.2 Eq.(8)), the order of signing is not important.

3.4. Problems when Extending to More Keys

To include more signing parties, more keys need to be generated. For example three parties need three keys (*d1,d2,d3*) and one public key *e*.
The first signature can be generated as in Sec.3.3.

$$psig1 = RSA(M, d1) \quad (13)$$

The second signer still needs to verify the signature (the first could have lied, and signed a manipulated message). The only knowledge is *psig1*, as well as the local secret *d2* and the public key *e*. However at this stage the unknown key *d3* of the third signer is needed for verification:

$$hash(M) \stackrel{?}{=} RSA(psig1, d2.d3.e) \quad (14)$$

doi: 10.2313/NET-2019-10-1_02

For more signing parties, even more secrete keys of other participants are needed. Without verification a signature should not be signed, since its integrity cannot be proven. With this "blind" signing, only the first and last signer can verify the signature. All intermediate signers can not contribute to the correctness (or truthfulness) of the signature.
Another alternative would be each signer adding their own signature to the document. However this defies the requirements of Sec.3, a client should only need to verify a single signature with a single public key. Additionally, for larger numbers of n signers a large overhead in signature length (n times more signatures) is generated. Needless to say that the verifier needs to check all n signatures, requiring the trusted knowledge of all n public keys.

3.5. Multi-Signature Schemes

An often presented solution are identity-based multi-signature systems. Here the signature is generated by using the identities of the signing parties. While this is an important solution to decrease the amount of public keys each participant needs to know (while maintaining security) and the complexion of signature verification [3] [4], these systems are not efficient at solving the problem of this paper. Especially since the verification process needs to be adapted to the signing algorithm.
However "A review of multisignatures based on RSA" [5] presents some schemes that could be adapted to fit the use case we need. However most of the presented ciphers require special prerequisites. As presented in [6], key generation is possible without any participant knowing the master key.

4. Example of Multi Signature for Certificate Authorities

In this section the system of Sec.3.3 is applied to a dual signature scheme for Certificate Authorities.
For sake of simplicity, a *website* will be *signed* resulting in a *certificate*. The website includes the full request for a certificate (used public key, website-operator, contact information, etc.).
Certificate Authorities (CAs) are tasked with verifying websites and signing their certificate. However the CAs use a single key pair to sign a record. With multi-key ciphers (and multiple signing keys) the risks of miss signings, or unauthorised signings could be reduced. It is assumed a website has been cleared for signing. Under normal circumstances, the CAs signing key would be used to generate the certificate. This can result in a number of problems. The signature will be secure, however the website might not have been cleared for signing.These errors are induced by the staff at the CA. And the easiest way to reduce these risks, is to establish a second (or even more) pair of eyes. A miss signing of one employee may be noticed by a co-worker. Additionally the responsibility (and the trust) for a certificate is evenly spread amongst several employees of the CA. A better way to handle the final signing of the website's certificate would be as follows (see Fig.3)

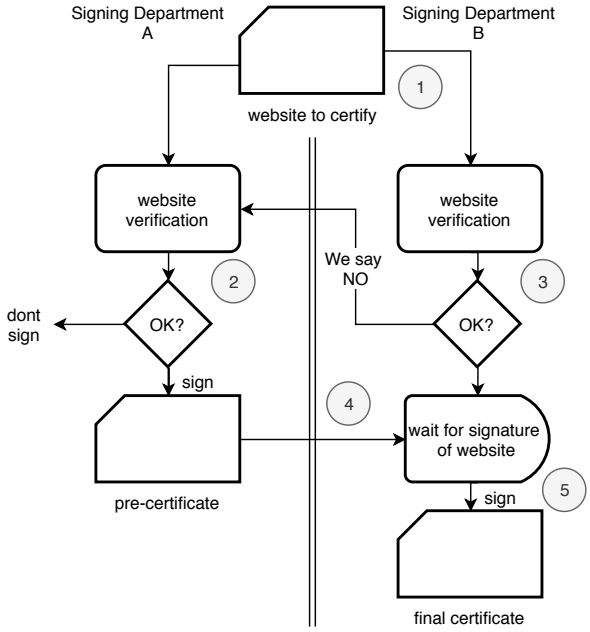


Figure 3: Basic RSA multi-signature scheme for CA

1) The website information (public key, owner, domain, etc.) is handed to two different departments. For sake of simplicity, one party is called *primary* signer. However, the signing order is irrelevant. Both departments only know one (different) key part of the master signing key.
2) The website is verified by the signers. This is done as if there was no second party involved. As a result this step can be taken in parallel: This will increase workload, but issuing time for a certificate remains the same (as with one signer).
3) Should the verification fail, the other department needs to be notified, in order to stop the signing process. Additionally the certificate will obviously not be signed with the local key.
4) If the primary signer trusts the website, it may issue a pre-certificate (using the secret key part) and send it to the second department.
5) Since (for two signers) the second party can fully decode the pre-certificate, it can be "re-verified". Due to working in parallel, at this point the second department probably is in the process of verifying said website their self (between stages 2 and 3). Should the website pass, signing the pre-certificate results in the final certificate.

In the given example (Fig.3) the primary signer will finish verification before the other department. However there are only two steps, where information is exchanged. On the one hand the website could not be verified (step 3)). In this case the first department to fail will contact the other one and no certificate is issued. Or everything is fine and one party has issued the pre-certificate. In step 4) the signer should first check if there is already a pre-certificate of the website, if there is they were slower than the other depratment and need to skip to step 5).
In this "two person signing" scheme, the two involved departments must have been supplied with the two different key parts. This needs to be done every time a new

doi: 10.2313/NET-2019-10-1_02

signing key is generated. Effectively the company is split into two parts, both need to work together to issue a single certificate. Spreading this further two CAs could work together to generate a "two CA trusted certificate". With intelligent use of multiple keys spread across CAs, this system would distribute trust to more than one company for each certificate.

5. Secret Sharing

The most used system of secret sharing (and the most used scheme) was invented by Adi Shamir in 1979 [7]. The basic idea is that a secret (in our case the private master key) is split into multiple parts. In [8] the system of splitting a key into multiple parts is already explored. Although this is not a multi key cipher, the possibility to split (or share) a key secretly is of fundamental need for multi-key ciphers. But instead of using the partial keys in a multi key cipher, other options are also possible.
The master key can be split (by simply using XOR operations) and the parts distributed. A central trusted entity then collects these partial keys, reconstructs the "master key" and signs the document. Obviously the central authority is in possession of the master key (or can read the key when all partial keys are entered). While this approach is heavily utilised for secure key storage [8], the partial knowledge forces the signers to rely on a central entity for signing. If all participants are part of the same company this might even be wished for, and there is already a "trusted" master entity (e.g. the boss, or board of directors). Compared to multi-key schemes this approach lacks security, since it returns to a single entity, that may not be compromised, and reintroduces the problem of "trust".
By using the properties of secret multi party computation, the knowledge of the master key can be circumvented. [9] describes a program, that computes a (public) RSA signature using the input of secret master key parts. This approach bridges the gap of limited trust and good usability. Here the central trusted entity is replaced by cryptographic primitives. All participants compute a RSA signature only with their local knowledge, by exchanging additional information (that carries no readable secrets), the final product will be the correct signature (with no one actually knowing more than before).
A new opportunity of strongly integrating secret sharing are (t,n) threshold schemes [10]. Instead of all n (n should be > 2) entities with a partial key, only a certain subgroup of $t < n$ are needed to successfully compute a solution. While this may not be a requirement for signatures, it increases the versatility of a signature scheme. For example multiple departments of a company can generate a signature. With this scheme, only 4 (out of 8) are needed for a valid signature. This reduces workload (only 4 departments must verify a website) and any 4 departments can generate a signature together (making load balancing across all work groups easier).
A possible danger of secret sharing are dishonest parties. By sharing faked information, many secret schemes may be defeated (the system won't generate the desired result and even secret information may leak). To ensure that all participants are using the correct algorithms and do not try to manipulate the scheme, verifiable secret sharing (VSS) [11] is needed. These algorithms enforce a correct utilisation of the scheme for all participants. If not trusting another participant, the integrity of their computation needs to be enforced this way.

6. Conclusion

The implementations of multi-signature systems are complicated. While they are feasible and can be used, the setup for signing a document with multiple signers is too difficult for the gain. As detailed in Sec.3.3, two signers can easily collaborate in creating a valid signature. Approaches requiring more participants are often required in a regulated environment. It is fair to assume that all entities are part of the same organisation, or at least know each other and have secure means of communication. In other words a trusted third party or centralised authority can be established (in fact [1] and schemes from [5] require such an authority). At this point the authority can also wait until all signers verified the document and then sign using a single master key.
The approach of incorporating secret sharing algorithms can bring back cryptographic security, while still spreading the signature process over multiple signers. Still such schemes are not straight forward and require trust or strong (and complex) cryptographic primitives.
In the end, the trade-off between risk of missignings (as well as security) and the requirements for implementing these systems needs to be considered. As long as all signing is done within the same entity, policies and management might be able to enforce better and less restraining signing processes. On the other hand in an internet driven world and the rise of crowd sourcing, multi key signature systems can be a first step towards distributed signing. The need for multi key signing in IoT systems incorporating "smart contracts" is also a pressing matter (here identity based signing (Sec.3.5) is heavily used).

References

[1] C. Boyd, "Some application of multi key ciphers," *Advances in Cryptology - EUROCRYPT '88*, 1988.

[2] B. Kaliski, J. Jonsson, and A. Rusch, "Rfc 8017," November 2016, [Online; accessed 07-04-2019].

[3] M. Bellare and G. Neven, "Identity-based multi-signatures from rsa," *Cryptographers' Track at the RSA Conference*, 2007.

[4] A. Bagherzandi and S. Jarecki, "Identity-based aggregate and multi-signature schemes based on rsa," *Public Key Cryptography – PKC 2010*, 2010.

[5] R. Duràn Dìaz and et al., "A review of multisignatures based on rsa," 2010.

[6] D. Boneh and M. Franklin, "Efficient generation of shared rsa keys," *Advances in Cryptology — CRYPTO '97*, pp. 425–439, 1997.

[7] A. Shamir, "How to share a secret," *Communications of the ACM. 22 (11)*, vol. 22, p. 612–613, 11 1979.

[8] G. Blakley, "Safeguarding cryptographic keys," *Managing Requirements Knowledge, International Workshop on (AFIPS)*, 1979.

[9] A. Mauland, "Realizing distributed rsa using secure multiparty computations," July 2009.

[10] S. Tang, "Simple secret sharing and threshold rsa signature schemes," *Journal of Information and Computational Science*, vol. 1, pp. 259–262, 12 2004.

[11] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," *Advances in Cryptology — CRYPTO '91*, pp. 129–140, 1992.

Routing in Information Centric Networks

Maximilian-Dominik Robl, Stefan Liebald*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga94kow@mytum.de, liebald@net.in.tum.de

***Abstract*—ICN is a new approach to change the current Internet architecture. In terms of locating and retrieving it differs a lot from the IP based internet. This paper gives n insight into the architecture of ICN. Also, this paper provides explanations of routing and naming in two examples of ICNs, NDN and NetInf. At the end of this paper, both approaches will be compared , especially in terms of routing and naming.**

***Index Terms*—software-defined networks, named data network, network of information, information centric network, routing**

1. Introduction

The current Internet architecture is based on the IP protocol. In IP a client is building a connection to a server, through this tunnel the publisher can send the requested data. This approach of connecting is similar to the telephone service where two persons are connected to be able to talk with each other. The IP architecture is from the year 1981 and oriented on a client to client communication. Over the last decades, the internet shifted from the usage of two clients communicating with a few users to a worldwide interconnected network where everyone is searching for specific data and not user.
With services like YouTube, Google, Netflix, Spotify, and co. many users request the same data from the same source, e.g. a live stream 100.000 people are watching.
Information centric networking (ICN) is the idea to replace IP with an architecture covering today's use cases of the internet. Providing an architecture that provides a client to data network. ICN attempts to create an architecture that doesn't need to search for the publisher but simply for the name of the needed data.
This paper focuses on routing names of the requested data and its naming, which makes this possible. For this we give you a brief introduction of ICN, especially naming and routing. This paper provides the basics of ICNs and explains the architectures NDN and NetInf based on this knowledge.

2. Routing Difficulties in ICN

The main idea of ICN is routing by the names of the requested data instead of routing by the addresses of the hosts [1]. Routing in general has difficulties. For ICN some of these difficulties are naming of the data (section 2.1) and occurring packet loss. If you download a file you download many packets. It can be enough that one byte is corrupted in your packet in any routing point so that you have to download this packet again. The problem is that instead of just going back to the last healthy file it instead the routing goes completely back to the server. This problem can be better managed with an ICN architecture, since most ICN architectures provide caching for their network nodes. Caching in this case provides the option to retrieve the lost packet from the last network node instead of the complete network path to the host.

2.1. Naming

An ICN architecture needs a suiting naming scheme, because it's fundamentally for the routing. Routing resolves the request with the name of the data until a node matching this name. Naming has some difficulties to regard. The names of the data in the network must be globally unique. [2] If the user wants to resolve this data the name must be clearly determinable to ensure that the user gets exactly the requested data. Another difficulty is the forwarding strategy in combination with the naming. For some architectures like NDN the forwarding strategy depends on the naming. Naming can also be human-readable or not. This is a trade-off that an architecture or the software engineer has to determine. Names in ICNs must be location independent. [3] This means that the names don't change with the environment they are saved. Names in ICNs mostly follow two naming schemes, first a structured or hierarchical naming scheme and second a structure-less or flat naming scheme. There is also a hybrid naming scheme which combines both naming schemes. Hierarchical names have a structure that can be interpreted similar to directories in operating systems. Flat names are missing structure.

2.2. Routing and Forwarding

Routing is one of the most important parts of an ICN since it significantly distinguishes from the current Internet architecture. Routing highly depends on the naming of the given ICN approach. In this paper, we discuss two routing schemes, name-based routing, and name resolution. Name-based routing works by using the name of the requested data to resolve the next hop in the network. Name resolution mostly works with a name resolution service, which returns some possible next hops in the network based on the committed name. A name

doi: 10.2313/NET-2019-10-1_03

resolution service can have different approaches [4], [5]. Routing consists of two main steps, locate the data by forwarding request and sending the requested data back to the subscriber. Every architecture can freely choose the way to design these steps. Especially the second step could still use an IP protocol since in ICNs forwarding the request should be based on names.

3. ICN Architectures

3.1. NDN - named data networking

NDN first appearance was in a google tech talk in 2006 by V. Jacobson. NDNs routes are lying in the earlier project content centric network which also was accompanied by V. Jacobson [6]. As an ICN paradigm, NDN takes a prominent role within the broader field of all ICN architectures. ICN is using various networking technologies below the waist for connectivity, including, but not limited to, IP [7]. Furthermore, the NDN architecture can be an independent routing architecture but includes the support to be built on top of IP for better integration in today's internet [8].
An important part of the architecture is the hierarchical namespace that NDN provides. It helps with name-based routing through its URL like structure [9]. The rough routing works by a client requesting data by sending interest packets that include names of the desired data. The NDN network then routes the interest packet forward until a node in the network that holds a copy of the requested data is found. The data packet is then sent back by this node. NDN as like most ICNs supports caching of data [10]. This is essential to give the routing paradigm the option to just send back a cached copy instead of always resolving to the original publisher. Also, the data objects should be independent from the location they are cached, but due to the idea of ICN routing by names, this applies for NDN.

3.1.1. Naming. Names in the NDN architecture are organized hierarchical. The names have a tree-like structure similar to URL, organized similarly to the directories in operating systems. E.g. Michi has a directory which contains projects and in this directory are another two directories raspberry pi and Arduino. In NDN we can request all data in these directories by using the prefix /Michi/projects/raspberrypi if Michi chose this organizational structure.
Names in NDN can be human readable like in the example but doesn't have to. Each component can be every format, a hash value would also be possible. It's a trade-off between human-readable and length of the name. The decision is to be made when implementing an NDN network. A human-readable name can have the advantage for the client to keep track when reading the messages, but also can lead to undesirable overhead. Also, global uniqueness is a necessary requirement which will be affected by the decision of the trade-off.
NDN uses a named-based forwarding design where forwarding is dependent on the naming structure. For resolving the names NDN uses longest prefix matching. E.g I'm searching for the name /Michi/project in the network. The forwarding protocol would first search for the prefix /Michi/, then for the prefix /project/ and would resolve to the name that fulfills the most prefixes. Important is that it will just route to identical prefixes. In the case of NDN, there is some specialty if I search for /Michi/projects the network will route for this name and return every data which shares exactly this prefix like /Michi/projects/raspberrypi and /Michi/projects/arduino. The names in NDN are strings with a flexible length. This gives us many possibilities to name our objects ensuring less overlapping of names and thus likely securing names to be globally unique. NDN is using pending interest tables storing object names and their location. Due to the flexible names they can easily get very large, resulting in a slower look up and overall network [11], [12].

3.1.2. Routing and Forwarding. Routing in NDN works with forwarding packets containing interests or data in the network. An interest is a request message forwarded as a packet. E.g. a client requests an information object with the name "/Michi/projects/" this would be sent as interest and the network would return a packet consisting of the data. The routing itself is resolved with longest prefix matching in the network. An item is found if the interests name exactly consists of an arrived node or if a prefix of the interests name consists.
NDN network architecture consists of special routers named content routers (CR). The content router extends a common router by providing three data structures, the forwarding table (FIB), the pending content store (CS) and the interest table (PIT). The forwarding table saves all the data to forward an interest. This is done hop-by-hop, the FIB can only forward an interest one step to the next router. This structure at least holds a column with names and one with fitting routing points. The content store is the cache of the router. The router can cache data traveling through it. The pending interest table saves all incoming interests. Only active interests which are not currently resolved are saved. If the interest found it's requested data and returns to the CR it will be deleted from the table. If a client requests a name that is already in the list it will be added in the PIT but not forwarded. The usage of a PIT in combination with the CS allows several people requesting data without resolving it several times. This increases the speed of the network and decreases the workload on the host because the host has to deal with fewer requests.
Figure 1 is an example of the routing an interest that is requested [7]. The routing consists of locating the data and returning the data to the client.
Beforehand the network starts with an empty FIB in the very first start. Forwarding is impossible in NDN with an empty FIB. The CRs must first set up the FIBs by searching for other CRs and information about the network. This problem is called "bootstrapping" and is used in today's internet, e.g. the ARP protocol is a way to resolve to bootstrap in a local network. NDN supports different protocols one is OSPF [6].
Step 1 to 3 in figure 1 shows the sending of an interests message for the data named **/aueb.gr/ai/new.htm** to the network and locating the data. In the first step, the clients send his interest packet to the closest CR. If a CR is obtaining an interest it always executes the following steps.

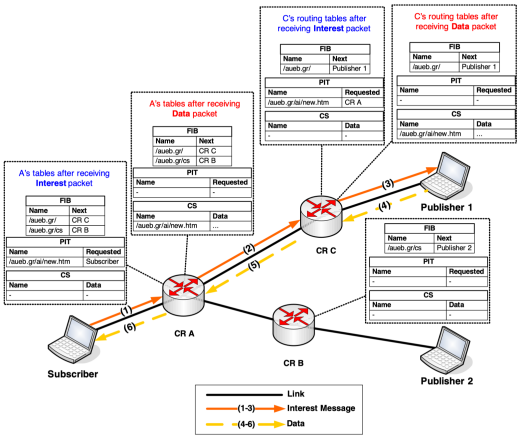
First, the CR checks it's CS containing an item with the exact name. If the item is found the CR just sends back a packet containing the data. If the item isn't in the CS the CR starts checking its PIT. In the PIT the CR checks if a request of the item already arrived independently from the requester's ID. If already a request for the exact item exists the CR aggregates both requests in the PIT and waits for the returning interest. If there is also no entry in the PIT the CR starts checking its FIB. Checking the FIB is done by using longest prefix match, e.g. the FIB entries for the following interest are /aueb.gr/ for CR2 and /aueb.gr/ai/ for CR3 the interest message would be forwarded to router CR3 because more prefixes match. This just applies if all prefixes are given a match. If there is just one prefix existing in the name of the FIB, but not in the interesting name the entry will be ignored.
In step 3 the CR finds the name of the interest in it's CS. Step 4-6 shows the retrieving of the data to the subscriber. In step 4 the current CR sends back the data to the last CR that has sent the interest. In the following steps, the CRs check their PIT every time they receive a data packet. The data is sent back to every client that requested the data. The CR checks all entries for the data objects name, sends the item to all subscribers linked to that name and deletes the interests in the PIT. NDN supports different types of caching strategies, thus the CR has the option to cache the data in every traveled CRs content store [10].

3.2. NetInf - Network of Information

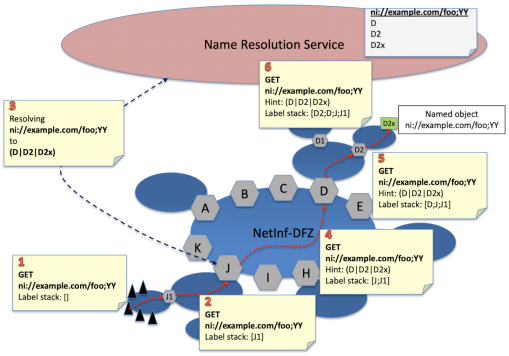
The network of information paradigm is one of the projects funded by the fp7 program of the EU and also part of the funded SAIL program. The principle of this paradigm is like in the most ICNs that the first order is accessing information via named data objects (NDO). NDOs are split into two parts one part is the name in a common format and the other part is the actual object in a common data structure [14].
This paradigm is said to have high support for migration. One part is playing the convergence layers (CL) which help the NetInf to be built on top of an existing routing or forwarding technology. Convergence layers are placed between the two layers and help them to communicate with each other. This helps to support a broad variety of different implementations and is also the idea of the creator, to create a paradigm which is very open in its implementation. Another way to achieve this is for the nodes to focus on minimal common node requirements to also be broadly applied to different types of networks. There is one naming format that all nodes understand and one format for representing the NDOs and optional metadata [13].
NetInf has its own protocol which is based on a few different messages. These messages are GET, PUBLISH and SEARCH. The GET message is used if I exactly know the name of the requested item, the PUBLISH message is used to advertise my information object and the SEARCH message is used to find an item by using keywords. With the SEARCH message, NetInf supports searching for information objects with keywords. NetInf nodes can implement the same request and response forwarding logic, transport and caching strategies for different networks that they are attached to [13]. Due to the ICN architecture and routing by names the routing is location independent and late-binding is supported, which results in the capability of caching information objects in the nodes of the network. NetInf supports on-path and off-path caching. The architecture of NetInf combines some design elements that are present in the NDN [6] and the PURSUIT [15] architecture. E.g. the possibility of routing on object names like in NDN and the idea of using a name resolution service like in PURSUIT. This will be described more precisely in section 3.3.2.

3.2.1. Naming. The namespace in NetInf is flat-ish. This means NDO names in NetInf can be flat, but names can also contain a hierarchy in their authority part. The authority part comes from the URI structure that names in NetInf fulfill. This structure was set by the SAIL project (the project with all papers and deliverables can you find here: https://sail-project.eu/deliverables/index.html) and has been registered as permanent URI schemes. A name in NetInf could be named like **ni://example.com/foo;YY** Considering the comparison of names NetInf names are flat [13].
An advantage is that a flat namespace provides better name persistent, due to its independence from organizational structure. If my object name in a hierarchical namespace is something like /de/user/data and I change this structure the name should be changed as well. This case won't happen with a flat namespace. A flat namespace also has the advantage of separating tussle over trademarks from unique data naming [13]. Uniqueness can be a problem in a hierarchical structure if two users are quite similar with also a similar organizational structure. With a flat namespace which is based on hashes, the naming can rely on statical uniqueness. In the case of a rare name collision, this can be handled as an error by the NRS. A disadvantage of a flat namespace can be the capability of aggregating names based on a hierarchical name. Yet naming in NetInf isn't completely flat, in the case of routing the names can be considered to be hierarchical. So it comes that in terms of routing NetInf supports name-based routing as well as naming resolution, which will be part of section 3.3.2. For name-based routing, it is also supported to use longest prefix match like in NDN.
Regarding naming scheme NetInf is using a common naming scheme which supports multiple "pluggable" cryptographic algorithms and representations [13]. The SAIL project registered two URI naming schemes that were designed for the NetInf paradigm and they are available to use for this architecture. The two registered naming schemes are "ni" and "nih". The scheme "ni" allows the inclusion of hashes in URIs in a structured manner [13]. On the other hand, "nih" is derived from "ni" by removing all optional features and ensuring the remaining structure was unambiguous when spoken [13].

3.2.2. Routing and Forwarding. As mentioned earlier the NetInf protocol is based on messages. These messages are used for request and response forwarding. As a small recap the messages are GET, SEARCH and PUBLISH, all of these also have a response part GET-RESP, SEARCH-RESP, and PUBLISH-RESP that are described in the protocol of NetInf. The routing of the message requires



(a) Figure 1: The NDN architecture. CR stands for Content Router, FIB for Forwarding Information Base, PIT for Pending Interest Table, CS for Content Store. [7]



(b) Figure 2: NetInf inter-domain scenario [13]

routing information to decide how to forward on each hop. The GET request is routing by the name of the requested NDO.

In NetInf name-based routing and name resolution is supported, which results in the possibility of a hybrid approach where either scheme can be freely chosen. Name resolution in NetInf works with routing hints. Routing in NetInf supports a component which is called routing hint. Routing hints indicate where to find copies of the object [13]. Routing hints are locators for lower layer hosts. The routing in NetInf forwards and resolves the request for objects as well as response messages. Here I also separate the routing in three different parts 1) bootstrapping 2) locate the NDO and 3) Return the information object. NetInf supports a big network like IP which exists of each other different network. All these networks also can have different routing requirements and thus need different routing protocols.

NetInf uses a hybrid request routing/forwarding scheme, which combines the possibility of using a name-based routing and name resolving. This is implemented by integrating pure name-based routing with name resolution aspects. That means that the routing paradigm tries to use name-based routing to forward the request and switches to a name resolution service if it can't find a hint for forwarding the next hop. Routers in a NetInf network also support a data structured called label stack similar to the PIT in the NDN architecture. The label stack stacks every name of the involved routers to easily travel back the route for the response message afterward. The name-based routing can use a pattern matching or as in NDN a prefix-matching approach. The NRS is then the system with a freely chosen algorithm that returns a set of routing hints.

For illustrating the routing scheme it needs a network to run in. In NetInf the creators assumed a network which is quite similar to today's internet. It is expected to have one global network which is expected of using just one routing scheme. This global network consists of different edge domains binding their network with the global network (The global network here takes the part of being the DFZ). Every edge domain can internally decide on NetInf routing/forwarding, adapted to the domains need. The created network relies on the hybrid routing approach. If a client will now request the object **ni://example.com/foo;YY** in the network the routing consists of 6 steps (Figure 2):

- Step 1: The clients sends a GET message to it's the closest network with the content ni://example.com/foo;YY as the name of the NDO.
- Step 2: The request gets forwarded to the next node. This can be done by name-based routing. In every step, the last router will be saved in the label stack.
- Step 3: The node is lacking routing information. It consults an NRS and gets a set of routing hints back. These will be added to the GET message.
- Step 4: Following the set of routing hints by performing the next hop the request reaches the next node. The set can also be just one element or there could be several nodes in between which are forwarded hop by hop with help of the routing hints.
- Step 5: The current node belongs to a different provider and thus has it's own routing/forwarding scheme. Our set also possesses all the necessary nodes to route forward based on it.
- Step 6: The requested node reaches a node holding a copy of the requested NDO.

Returning the NDO is done by the node holding the NDO sending a RESPONSE message containing the NDO and the label stack. This message is then routed step by step dismantling the label stack. While routing backward NetInf allows caching the NDOs [13].

4. Architecures differences and conclusion

In this paper we summarize the ICN approach and two of its current architecture. Both architectures can be implemented completely independent to the current IP infrastructure, still, both architectures support to be built on top of an IP network, the NDN architecture with the option to be built on top of existing routing architectures and NetInf with its CLs.

In terms of routing NDN provides a hierarchical structure

doi: 10.2313/NET-2019-10-1_03

which has an enormous impact to its routing. NetInf has a flat-ish naming approach, which combines the hierarchical naming scheme with a flat one. Due to the flat-ish naming structure and hashing support, NetInf ensures global uniqueness with statistical uniqueness, while NDN uses its prefixes to ensure this global uniqueness.
The NDN naming approach though can due to its prefixes have very long names. Also the approach of routing with names create more entries to look up. With IP adresses a host had one adress and contained several data objects. In ICN we route for the data objects, which are in a larger number. This can result in the FIB and PIT to be very large, consuming storage space and decrease the lookup speed. The NetInf architecture uses an NRS which tends to be the bottleneck of the network. Since the NRS will be called by several access points to request routing information and has to store more entries in the table than the NDN approach. Also the NRS is the location were some sort of table like the FIB and PIT is used to resolve the next hops for the request. And the NRS is potentially linked to a larger network than a CR, which can also lead to a larger table.
Both architecture support caching in their nodes, which in most cases can reduce the number of traveled nodes and thus the lookup speed, because the data can easily retrieved by these nodes. Also, packet loss can be handled way better in both architectures. If packet loss occurs the client can request the data from the last node holding a copy. In case of packet loss, the data packets mostly have to travel fewer nodes as in the first request making the network faster.

References

[1] M. Awais and M. A. Shah, "Information-centric networking: a review on futuristic networks," in *2017 23rd International Conference on Automation and Computing (ICAC)*. IEEE, 2017, pp. 1–5.

[2] M. S. Akbar, K. A. Khaliq, R. N. B. Rais, and A. Qayyum, "Information-centric networks: Categorizations, challenges, and classifications," in *2014 23rd Wireless and Optical Communication Conference (WOCC)*. IEEE, 2014, pp. 1–5.

[3] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Communications Magazine*, vol. 50, no. 12, pp. 44–53, 2012.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[5] X. Jiang, J. Bi, G. Nan, and Z. Li, "A survey on information-centric networking: rationales, designs and debates," *China Communications*, vol. 12, no. 7, pp. 1–12, 2015.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[7] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[8] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[9] H. Yuan, T. Song, and P. Crowley, "Scalable ndn forwarding: Concepts, issues and principles," in *2012 21st International Conference on computer communications and networks (ICCCN)*. IEEE, 2012, pp. 1–9.

[10] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named data networking of things," in *2016 IEEE first international conference on internet-of-things design and implementation (IoTDI)*. IEEE, 2016, pp. 117–128.

[11] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *Named Data Networking Project, Technical Report NDN-0034*, 2015.

[12] N. L. Van Adrichem and F. A. Kuipers, "Globally accessible names in named data networking," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 345–350.

[13] D. Kutsher *et al.*, "Final netinf architecture," https://sail-project.eu/deliverables/index.html, 2013.

[14] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (netinf)–an information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721–735, 2013.

[15] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in *International Conference on Broadband Communications, Networks and Systems*. Springer, 2010, pp. 1–13.

Surveying the depth of user behavior profiling in mobile networks

Dominik Spörle, Marton Kajo*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: d.spoerle@tum.de, kajo@net.in.tum.de

***Abstract*—CDR (Call Data Records) allow network operators to collect location information, as well as other application level data about users of mobile devices. With various machine learning techniques it is possible to extract information to deduct general, or also very personal insights into the user's behavior. This survey presents various research topics working with CDR data, such as the evaluation of the mobility or social interactions of the users. This information is processed by using data mining methods, which yield patterns used to get insights in the user behavior.**

***Index Terms*—call data records, data mining, machine learning, mobility patterns**

1. Introduction

Mobile phones are popular devices for decades and used by a lot of most people daily. According to [1] in 2019 the number of mobile phone users is forecast to reach 4.68 billion. This entails a big amount of data. A CDR is a data structure created by network providers and stores relevant information about the telephonic activity. A CDR contains usually a spatial and a temporal resolution, which enables researches on the behavior of users by analyzing CDR data.
CDR data is often compared to GPS data, because it can track users in time and geographical space. It is important to note that GPS data is more precise and delivers more information about the movement of users, but CDR data is available since longer time and in a higher amount. Furthermore, CDR data can also contain information about the user's social interactions.
By analyzing patterns extracted from CDR data there are various areas to apply this knowledge. The understanding of mobility patterns yields insights into crowd analysis, population displacement, urban planning, network design, traffic management, targeted marketing, tourists movement or disease spreading. By analyzing the social interactions of calls between users it is possible to discover relationships, calling patterns or the social attributes of users. This information might help for example in areas like criminology to detect the social networks of criminals.
This paper presents and describes various techniques of creating patterns about the user behavior, while also highlighting the importance and usefulness of CDR data for many different areas by selected examples.
In Section 2 the processing of CDR data is described with different approaches, primarily showing methods to create movement trajectories of users by interpolating the spatial information of CDRs. In Section 3 different applications of CDR data is presented.

2. Handling & processing of CDR Data

To use CDR data to get insights in the behavior of users the data must be processed to get a movement trajectory of the users. Since CDRs only contain data of calls or texting activities of users, the accuracy of this data along the spatial and temporal dimension is limited, often referred to as spatiotemporally sparse data. To mitigate the sparsity of CDR data there exist several approaches of data completion, which try to fill the spatiotemporal gaps and derive movement trajectories from it. To do this the position of a user between the respective phone calls or sent text messages has to be estimated. This is done by applying movement models to the data with the use of machine learning techniques and data analysis. This is explained in the following sections.
Before the data of CDRs can be used for research, it is necessary to draw attention to privacy issues associated with CDR data. To work with it, the data must be anonymized to prevent that personal data can be inferred from it. Some of the approaches to do that are presented in Section 2.3.

2.1. Call Data Records

A call data record, or short CDR, contains in general the respective time of the call and the ID of the user. Also, it has an entry for the location, which is saved as the ID of the prefecture from where the call occurred. This information can be extended by appending data like the day of the week, the time of the previous call, etc. The structure of a CDR may depend on the communication service provider publishing the dataset.

2.2. Mobile Positioning based on CDR data

There are several approaches to refine CDR data and complete mobile positioning of users. As a first approach, the position of the user in between CDR events can be refined using a probabilistic model. In [2] an Inter-Call Mobility model (ICM) is introduced, which is based on a finite Gaussian mixture model.
The ICM model represents a spatiotemporal probability distribution of the location of a user between two consecutive CDR events. It relies on the Gaussian mixture model (GMM), which is a weighted sum of Gaussian Probability

doi: 10.2313/NET-2019-10-1_04

Density Functions (PDF). These functions represent the probability of finding a user at a position (x, y) at time t. The GMM is defined by the vector of all unknown parameters θ. Those parameters are estimated by the Expectation-Maximization algorithm [3], which performs a maximum-likelihood estimation. The number of components K in a Gaussian mixture is selected using criteria that combine the parameters estimation and a penalty that tries to prevent overfitting of the model. An initial estimate of the parameters θ is given by $K - means$ clustering.
In [2] the GMM is fitted to inter-call trajectories created during the data processing of the CDR data and the ICM model is defined. Applying the ICM model to two consecutive CDR events a probabilistic distribution of the user's position between the calls can be created. The the kernel density estimation of spatiotemporal probability distribution of user's inter-call movement is shown in Figure 1.

A second approach presented in [4], [5] is to create

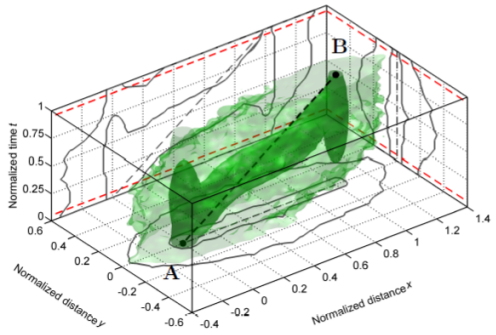


Figure 1: ICM model: probabilistic distribution of the user's position between the calls A and B, [2]

a model out of studies and analyzes of user's behavior. This is done by considering a user to stay more likely at certain locations, like his home or work place. The authors of this work separate the activities of users between a nighttime and a daytime period. To capture the locations for home and workplace the data is separated in those two periods and significant places are extracted. This is done by considering the place where the majority of CDR events during daytime occur to be the work place of the user and the place during nighttime to be his home.
For both day- and nighttime, different techniques are used for the completion of inter-call localization. During the nighttime period the user position is set to his home location if the last CDR is within some fixed temporal home boundaries. This method is expanded by adapting the nighttime interval and the home boundary for each user on previous observations made of him. For the completion of CDR data during daytime three factors are categorized, which affect the temporal cell boundaries of the user. Those factors are event-related, long-term behavior, and location-related. Event-related factors relate to the metadata of the CDR, like the duration of a call, what may give some indication if the user is static or moving. An example of a long-term behavior factor could be the number of unique visited locations, which can be related to the long-term mobility of the user. Location-related factors can yield to indications how relevant different places are in the user's activity.
With this information a model is created with supervised machine learning techniques to estimate the corresponding temporal cell boundary. The model is generalized from a training set consisting of CDR entries. Approximations are made with the Gradient Boosted Regression Trees technique [6].
In a third approach [7] the coverage area provided by the mobile operator and the location of the mobile device within this area is estimated. Also, the type of movement is detected in order to differentiate between a moving and staying user. Then, a map-matching technique is used to match the resulting location to a road (if a moving user is detected) or a building (if a stationary user is detected). To do this, first, the coverage area is estimated using the Voronoi method [8] and is optimized by comparing it with collected GPS data. This is done by minimizing a penalty function based on the observations of the GPS data. The coverage function is minimized with the implementation of the L-BFGS-B algorithm [9]. To differentiate between a moving and a staying user an implementation of a Kalman filter [10] is used and a mobility model is defined and integrated into this algorithm. For the map-matching of an estimated position, for each point corresponding road segments are found and matched. Therefore, a detection of road segments in a certain radius around the point is implemented and for each segment candidate points are computed with an orthogonal projection. The best candidate point is selected using the haversine distance [11]. A result of this map-matching step is shown in Figure 2. In general, it can be observed that the presented

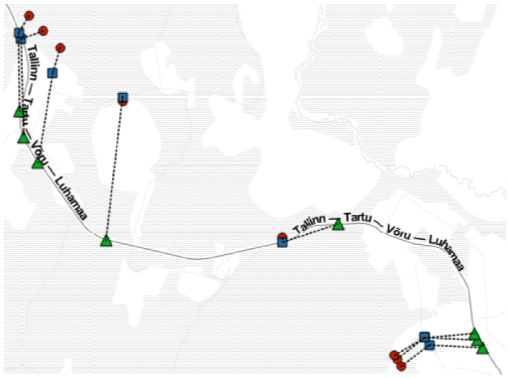


Figure 2: Map-Matching of CDRs: results of Kalman filter - red circles - estimated position, blue squares - map-matched position, green triangles - actual position, [7]

approaches yield to good results, if you consider that CDR data consist of sparse informations about the mobility of users. Estimated inter-call positions are not comparable to technologies like GPS, but they are precise enough to get useful insights into the behavior of users.

2.3. Anonymization of CDR data

As we can see in further chapters, CDR data can deliver personal information about users, which can cause privacy issues when working with it. It is challenging to find a good balance between the protection of the privacy of users and the utility of the data itself. Of course, published datasets do not contain any personal

information, but also anonymized data entails the possibility of re-identification of users. Traditional methods of anonymization are pseudo-anonymization (ID of the referenced user in the CDR is replaced with a code using cryptography), k-anonymity (trajectory of a user is hidden among $k-1$ other users with the same quasi-identifier) and spatial location cloaking (spatial noise is added to the data), but these methods are not efficient, because they either are not preventing re-identification or they impair the utility of the data.
An approach presented in [12] is to add time distortion instead of spatial distortion to the data. With a mechanism called Promesse the POIs (Points of interest) of users are hidden by smoothing the speed of the movements along the trajectories (see Figure 3). This approach is designed for mobility data in general, e.g. also GPS data, and is a good example of a method restricting the utility of data - especially if you apply it to CDR data. The protection of privacy is working, but one goal of CDR data processing is to find likely locations of users (POIs) and this approach is not very useful for this.
A more valuable anonymization method is described in

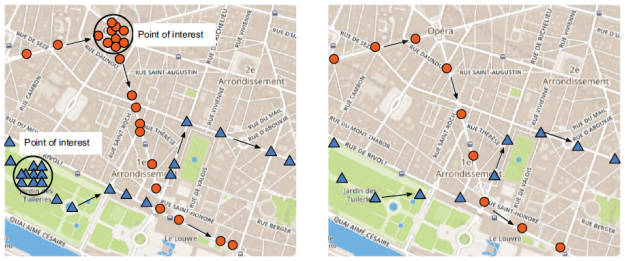


Figure 3: Overview of Promesse: smoothing the speed of movements - left: original dataset, right: after enforcing a constant speed, [12]

[13]. Here, the authors propose Differential Privacy (DP), a method that creates synthetic data out of the original one without any one-to-one correspondences between the two datasets. Their method DP-Star processes key factors and statistics learned from the original data and generates synthetic data out of it. This is done with 5 components of the system (which is shown in Figure 4). First an adaptive grid construction processes an effective discretization of the geographical location space of the dataset. With the trip distribution extraction, the correlation of the start and end point of a respective trajectory is kept. To mimic movements patterns of actual trajectories a mobility model is constructed, which is a Markov model. To estimate the route length of a synthetic trajectory a private median retrieval procedure is applied, which returns a noisy median of the trajectories. As last step the synthetic trajectory generation is processed in 5 separate steps: The start and end cells are generated by drawing a random sample from the trip distribution. The route length is determined by approximating it with an exponential distribution of the median lengths. The synthesizing of the trajectory as a sequence of cells is done with a random walk on the Markov mobility model. Finally, cells are converted to locations by randomly sampling a position with each cell. As a result a sequence of locations is the final trajectory.

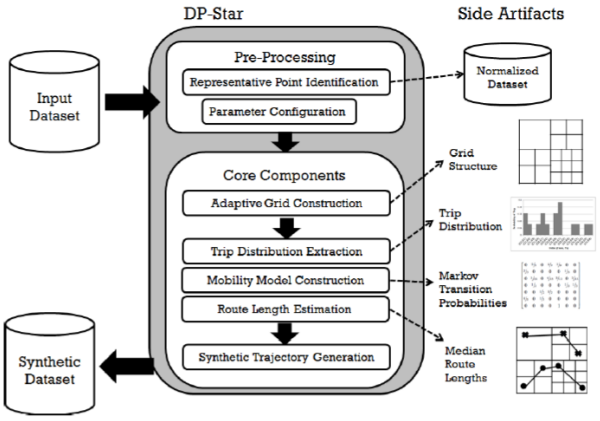


Figure 4: DP-Star architecture, [13]

A comparison of an original and a synthetic trajectory can be seen in the Figures 5 and 6.

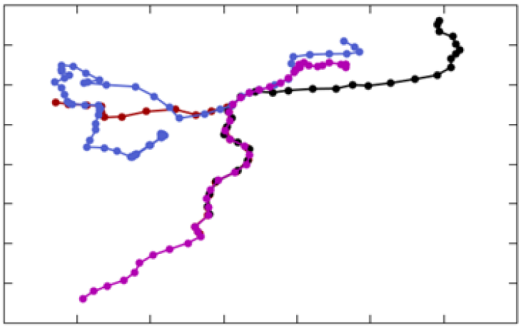


Figure 5: DP-Star: original CDR trajectory, [13]

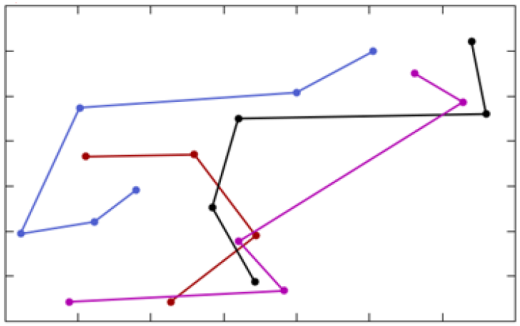


Figure 6: DP-Star: synthetic CDR trajectory, [13]

3. Insights

With the extraction of clusters from CDR data especially mobility patterns can give insights in the behavior of users. This can be used in a large variety of applications.

3.1. Data mining on CDRs

With the goal of characterizing users and extracting their behavior data mining analysis on CDR data is implemented. There are several different procedures for

doi: 10.2313/NET-2019-10-1_04

clustering and knowledge discovery. Often, they are based on analysis of the data itself, which yields to the definition of different subsets of features. For processing the CDR data can be structured as graphs, labeled sequences or sectioned vectors. The choice of the data structure can affect the results, it should be considered if for example topological information or computational resources are more important to the selected method.
A concrete data mining approach is presented in [14]. Here the cluster discovery is implemented with the LD-ABCD algorithm, which extracts separated clusters in the data and returns for each of the clusters the most appropriate local metric. This multi-agent algorithm is working on a weighted fully connected graph representing the data per agent. On each graph clusters are discovered by means of multiple Markovian random walks. These patterns are evaluated by a measure called Cluster Quality, which is dependent on the concept of graph conductance and the configuration of the dissimilarity measure. An agent might identify a set of similar clusters, which are merged to a meta-cluster. The output of the algorithm is a set of such meta-clusters. These clusters do not necessarily include the whole dataset. As a result, LD-ABCD is able to discover regularities and patterns among CDR data.

3.2. Analysis of tourism dynamics

In tourism data about the behavior is often created by analyzing interviews and surveys. In the area of big data social networks and economic datasets play a big role. Working with CDR data improves the quality of knowledge in various aspects of the tourism industry by extracting new indicators like, for example, the flow of tourists or profiles about different interests of tourists. These indicators can add value to the evaluation of touristic events and marketing strategies. In the approach presented in [15] indicators are mined from CDR data in the context of the country Andorra. As a result, the authors extract 6 different indicators by analyzing CDR data and comparing it to self-reported tourism surveys. The indicators found are:

- segment tourist flows (based on country of origin)
- new tourists and repeated tourists
- spatial distribution (based on country of origin)
- congestion
- revenues: gained by estimating the income of tourists in order to obtain the price of the mobile device used with the IMEI-TAC-code of the records
- tourists interest profiles: gained by comparing POIs of tourists with activities nearby respective cell towers

The authors then demonstrate in a case study how to add the value of those information to evaluations of tourism strategies by analyzing summer and winter events and tourist interests' profiles in Andorra. The results of those profiles can be seen in **Figure 7**.

3.3. Improved Quality of Experience with predictive models

For communication service providers (CSP) it is a big challenge to find patterns of customer behavior to improve

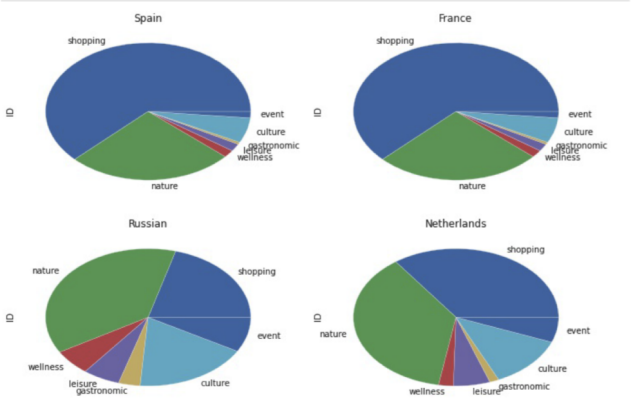


Figure 7: Tourism profiles: visualization of the nationality of tourists vs. their interests, [15]

the network and the customer's satisfaction. To assure a good level of Quality of Experience data management systems including Fault Management and Performance Management are required. To improve the performance of networks operators need analysis and diagnosis tools. Such a diagnosis tool is presented in [16]. This solution, which is called ARCD, Automatic Root Cause Diagnosis, is able to locate the root cause of network inefficiency. It uses logs which contain CDR data. Those CDRs are labeled either failed or successful and processed in several steps using equivalence class and graph computation. Finally an automated system diagnosing cellular networks based on the data collected from large-scale monitoring systems is implemented.
In [17] Customer Relationship Management (CRM) records are used to build a predictive model for customer churn (termination of the user's contract). To do that, CSP store customer transactions to discover patterns of customer behavior, which helps to find solutions to reduce their contract termination. CRM records contain contractual data. CDR data is required to complete it in order to predict churn. The approach is based on logistic regression [18] and random forest models [19].

3.4. Identifying criminal's behavior and social relations

Using CDR data it is possible to generate useful informations about the social relations and the behavior of users. Also, in the field of criminology these insights might be helpful to crack a criminal case. In [20] crime information and CDR data is combined to extract relationships and interactive patterns of criminal suspects.
This is done by implementing a knowledge graph analysis, which uses several graph traversal algorithms. The two datasets, CDR data and criminal cases, are both imported into a knowledge graph. The phone number of a single CDRs is implemented as nodes as well as the number of a criminal case. Edges are the call records itself, connecting phone nodes and the crime records connecting phones with related cases. The resulting knowledge graph is computed with a shortest path algorithm to discover shortest paths in the graph and thereby the contacting

doi: 10.2313/NET-2019-10-1_04

chains between users. Then the betweenness centrality algorithm is applied to the graph to extract the pivotal person in a social circle. With the Pagerank algorithm from Google (can be used as part of APOC library [21]) the importance of a node is calculated (as an example see Figure 8).
To detect different types of relationships a clustering model is created. To generate clusters features are defined based on the analysis of the respective data. In this approach features contain only the temporal component of CDRs, e.g. for example the duration of a call or the total number of calls during a specified time interval. With the features a Gaussian mixture model method computes the clustering model. As result 5 different clusters are extracted from the data, which contain unique characteristics. As an example, one of the resulting clusters has the properties that the contact of the users is kept for a longer period of time, they have a long holding time of their calls and the calls mostly occur in a working time and in relatively large numbers. This cluster reveals the relationship of the contacting users to be linked to their work. So, the model can extract the relationships and to separate these into possible categories like criminal confederates, working colleagues or close friends.

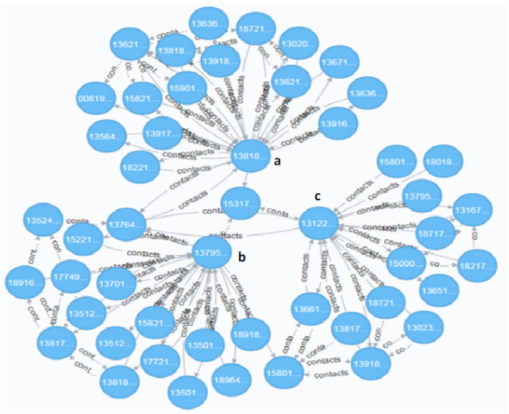


Figure 8: Page rank algorithm applied to a knowledge graph, [20]

4. Conclusion

It can be determined that CDR data can yield to useful information of the behavior of users. This can be used in a wide range of different topics. It can be concluded that especially the mobility patterns and the information about social interactions extracted by analysis of the data is valuable. Extracting information like the POIs of a user and his social relationships can be applied to improve research about human behavior and help to implement smart solutions for areas like marketing, urban transports, criminology, tourism and a lot more. To get good results working with CDR data it is often combined with other data, which can be geopositional or topic-related. Anonymization of CDR data is an important issue and current solutions do not face this issue in the way they should. On the other side approaches like DP-Star promise acceptable utility of the data and the protection of the privacy of users.

References

[1] Statista, "Number of mobile phone users worldwide from 2015 to 2020 (in billions) ," https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/, 2019, [Online; accessed 07-April-2019].

[2] M. Ficek and L. Kencl, "Inter-call mobility model: A spatio-temporal refinement of call data records using a gaussian mixture model," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 469–477.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[4] S. Hoteit, G. Chen, A. Viana, and M. Fiore, "Filling the gaps: On the completion of sparse call detail records for mobility analysis," in *Proceedings of the Eleventh ACM Workshop on Challenged Networks*. ACM, 2016, pp. 45–50.

[5] G. Chen, S. Hoteit, A. C. Viana, M. Fiore, and C. Sarraute, "Enriching sparse mobility information in call detail records," *Computer Communications*, vol. 122, pp. 44–58, 2018.

[6] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[7] A. HADACHI, Amnir; LIND, "Exploring a new model for mobile positioning based on cdr data of the cellular networks." *arXiv preprint arXiv:1902.09399*, 2019.

[8] Wikipedia, "Voronoi diagram," https://en.wikipedia.org/wiki/Voronoi_diagram, 2019, [Online; accessed 12-May-2019].

[9] J. L. Morales and J. Nocedal, "Remark on" algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization"." *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 7–1, 2011.

[10] F. Xiao, M. Song, X. Guo, and F. Ge, "Adaptive kalman filtering for target tracking," in *2016 IEEE/OES China Ocean Acoustics (COA)*. IEEE, 2016, pp. 1–5.

[11] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[12] V. Primault, S. B. Mokhtar, C. Lauradoux, and L. Brunie, "Time distortion anonymization for the publication of mobility data with high utility," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 539–546.

[13] M. E. Gursoy, L. Liu, S. Truex, and L. Yu, "Differentially private and utility preserving publication of trajectory data," *IEEE Transactions on Mobile Computing*, 2018.

[14] F. M. Bianchi, A. Rizzi, A. Sadeghian, and C. Moiso, "Identifying user habits through data mining on call data records," *Engineering Applications of Artificial Intelligence*, vol. 54, pp. 49–61, 2016.

[15] Y. Leng, A. Noriega, P. A. S., I. Winder, N. Lutz, and L. Alonso, "Analysis of tourism dynamics and special events through mobile phone metadata." *arXiv preprint arXiv:1610.08342*, 2016.

[16] M. Mdini, G. Simon, A. Blanc, and J. Lecoeuvre, "Arcd: a solution for root cause diagnosis in mobile networks," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 280–284.

[17] K. A. NESTOR, Dahj Muwawa Jean; OGUDO, "Practical implementation of machine learning and predictive analytics in cellular network transactions in real time." *International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD) IEEE, 2018. S. 1-10.*, 2018.

[18] P. Bühlmann and B. Yu, "Boosting with the l 2 loss: regression and classification," *Journal of the American Statistical Association*, vol. 98, no. 462, pp. 324–339, 2003.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] Y. FAN, T. YANG, G. JIANG, L. ZHU, and R. PENG, "Identifying criminals' interactive behavior and social relations through data mining on call detail records," *DEStech Transactions on Computer Science and Engineering*, no. aiea, 2017.

[21] neo4j, "APOC Labrary," https://neo4j.com/developer/neo4j-apoc/, 2019, [Online; accessed 12-May-2019].

Matrix Cryptography

Franziska Steinle, Jonas Jelten*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge34weg@mytum.de, jelten@net.in.tum.de

***Abstract*—In this paper the main topic is the Matrix cryptography. Matrix is a system that helps humans and machines to communicate over different ways. Matrix tries to be the main platform to communicate. Many existing platforms are missing the save encryption. This subject affects most people at the moment, because the issue of safe communication is a very important topic in view of the fact that many people use these services. Matrix provides safety with the two encryption algorithms Olm, for conversation between two, and Megolm, for group conversation. The paper concentrates on how these algorithms work.**

***Index Terms*—matrix, olm, megolm, encryption, double-ratchet algorithm, communication, iot, bots, webrtc, video-telephony, messaging apps**

1. Introduction

WhatsApp, Telegram, Slack, Line and many more are platforms we daily use to communicate with others. But if you want to organise something, often a problem will arise. Namely one or more of the concerned persons does not use the platform you want to use to communicate. This is because of the high fragmentation that exists in the communication business. Matrix is created to get rid of this problem. It should help you to reach all people, who are at least registered at one platform.

Another problem, which is present in our daily life, is the safety of our data. Recently data were stolen from Facebook [1] again and everyone considers, which platform should be used to guarantee that our personal information does not get stolen. Matrix provides safety, because firstly the data are not saved at one server, where many information can be stolen at once, but on many different servers. Also the severs save only the encrypted versions of the messages, the addressee and the sender are the only ones who can decrypt them.

As you can see these issues affect everyone who does not want to switch between platforms all the time. Also it is easy for developers to integrate existing platforms to the big network. Matrix tries to connect many platforms, to make modern communication easier, like we know it from using Email.

The main topic this paper handles is how the safety of the data is guaranteed. Mainly two algorithms are used, Olm and Megolm. It is described detailed how they work and given a little overview about the features Matrix provides.

The paper is grouped in three main parts. The first handles the topic, what Matrix is and what it can be used for. The second section describes the Olm algorithm, details the double-ratchet algorithm, which is the origin of Olm, the initialisation, the main algorithm and the difference of the two described methods. The last part handles Megolm, also with the initialisation and the main algorithm in detail.

Figure 1: The Matrix Logo [2]

2. Related work

The next section is about the work that is related to this paper. First we should remind you that the topic is a very new one. The developing of Matrix started in 2014 and there are a few to none papers about this topic.

But Matrix is not the first attempt to standardise online communication, there were others. But all failed and from the mistakes that were made the developers of Matrix tried to learn.

Because the topic is very new, mostly literature is used, that is provided by Matrix itself. Matrix is a open source project, so many information about the detailed encryption process is available. Even the code itself is public.

3. Matrix

At the following part we describe what Matrix is and what it is used for. Matrix is a decentralised communication network. Decentralised means it has no main server on which everything is saved, but the data is duplicated at every participating server. It supports encrypted one-to-one communication and also group messaging. Matrix also provides real-time synchronisation and the messages that are send in JSON format are saved on all participating servers. [2]

The one-to-one communication encryption is based on the double ratchet algorithm and is called Olm, whereas the group communication is encrypted with Megolm. Olm and Megolm will be described and explained later in this

doi: 10.2313/NET-2019-10-1_05

paper. These algorithms guarantee end-to-end encryption, which means that the messages are saved encrypted at the servers and only the addressee can decrypt them again. [2]

The development of Matrix started in 2014 with a team, who was employed by Amdocs to work on this project, and since 2017 they founded their own independent company, which is called New Vector. The main team, consisting of circa twelve people, is supported by many other developers. [3]

The servers are saving the history of the communication. When a client sends a message, it will first be added to the path at his own server and then sent to the other servers. There the message will be checked, whether the sending client is really him and if the client can transmit messages. If everything is correct, the message will be added to the server's history. It can happen that two or more clients send their messages at the same moment, then the history graph splits and when the concurrent situation ends, the paths are merged together again, like it is done in Git. Because of this handling the histories of the servers are always the same. [2]

3.1. Usage

Now we describe the many different ways the Matrix network can be used. It can be used to connect messaging apps like Telegram, WhatsApp and Slack. The network supports interoperable communication, so that not everyone needs the same application to chat. The usage can be compared to Email, because writing and receiving them is not connected to the program you use. Building new bridges from existing messaging applications to Matrix is easy. For example the link to Slack has fewer than 100 lines of code. Matrix can also be added as a chatroom to other Apps, who do not use any chats until now. Encryption, Emojis, file transfer and many more features are possible with Matrix. [2]

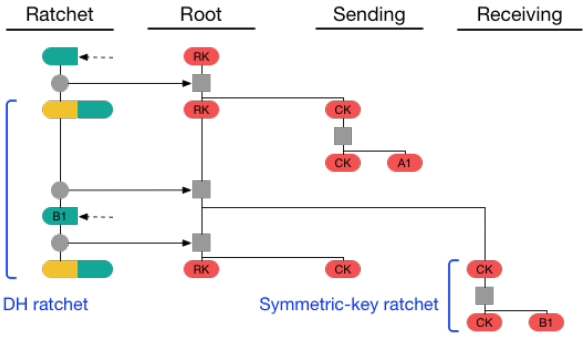
Matrix can be used in the Internet of Things (IoT), which is for example used in cars and drones. Matrix can connect different IoT silos and support them to communicate. The information gained from the silos can be published directly from the device under the user's control. Until now the fragmentation in the IoT is very high and Matrix can help to solve this problem. When developing a new device, the developers are also able to directly work with Matrix. [2]

Another usage is for Voice over IP and WebRTC, so phoning and video-telephony and many other things are possible. So far there is no standard protocol for this kind of communication. Matrix tries to become that, because it is build simple and familiar for Web developers, so they can integrate it easily to their Websites. It can be used in Apps too. [2]

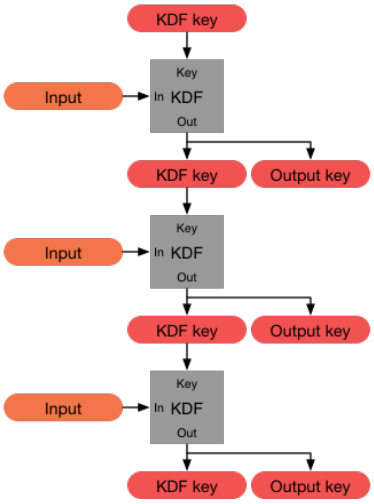
The last described way to use it, are bots. Bots must be developed for every platform separately, but with Matrix they only need to be programmed for one. [2]

3.2. Features

Now additional features of Matrix are shown that can be used, in a room. In a chatroom you can see, if the other users are online, typing or if they have already read your message. Also you can adjust how often the server informs you that new messages are available. The server can be searched to find old messages. Additionally the account data of every participant in a room is saved. [2]



(a) Chain at the double-ratchet algorithm [4]



(b) The KDF key chains [4]

Figure 2: Double-ratchet algorithm

4. Olm

At the following part we describe the Olm algorithm, which is based on the double-ratchet algorithm. Olm is used to guarantee end-to-end encryption in 1:1 communication.

4.1. Double-ratchet algorithm

At this section we look at some parts of the double-ratchet algorithm, which is the base of the Olm algorithm. It helps to understand the explanation of the Olm algorithm, which follows later in the paper.

Every message is encrypted with its own key, so hacking the system is harder. These keys are generated with KDF chains. These chains take a secret and random KDF key and input data and produce output data, which is then split in an output key and a new KDF key for

doi: 10.2313/NET-2019-10-1_05

TABLE 1: Olm Pre-key Message Tags

Name	Tag	Type
One-Time-Key	0x0A	String
Base-Key	0x12	String
Identity-Key	0x1A	String
Message	0x22	String

TABLE 2: Olm Normal Message Tags

Name	Tag	Type
Ratchet-Key	0x0A	String
Chain-Index	0x10	Integer
Cipher-Text	0x22	String

the next step in the chain with new input data. All clients have three chains, one for sending, one for receiving and a root chain. [4]

One part of the double-ratchet algorithm is the Diffie-Hellman ratchet. Every client has a Diffie-Hellman key pair with a public and private key. The sender of a message sends the public part at the beginning of the message and if the addressee does not know this key, he creates himself a new key pair. When a new key is generated, another output is also created. This is called a Diffie-Hellman ratchet step. The result of this algorithm is a constantly changing key pair. [4]

The output of the Diffie-Hellman algorithm is used to produce new sending and receiving chains, because it works as input for the root chain. The output from the root chain is then used as a new KDF key either for a new sending or a new receiving chain. [4]

The output data from these chains are used as message keys. The message keys from the sending chain are used to encrypt the message and the keys from the receiving chain to decrypt them. The inputs in these chains are constants. This is called the symmetric-key ratchet and it works because both participants start with the same Diffie-Hellman key pair and so all the following chains have the same outcomes. The only difference is that the sending and receiving chains are switched. [4]

4.2. Initialisation

Now it is shown how a room must be initialised to make Olm possible. At first one participant publishes the public part of his identity key and some single-use keys. The other participant takes the identity and one single-use key and builds his own single-use key. With the identity keys and the used single-use keys a shared secret is made using the Diffie-Hellman algorithm. This shared secret is then used to generate the first root key, the first chain key and a ratchet key. [5]

The next step is that the second participant sends a pre-key message to the first. Pre-key messages consist of a version byte, which is usually 'x\03' and payload bytes. The payload bytes have key-value format, in which the keys are encoded. The last three bits of every encoded key will show, if the following value is an integer or a string. Encoded strings have first a specific tag followed from his encoded length and then the string itself. Integers also have a tag followed by a byte, which saves the least significant bits from every Byte of the original integer. These tags for the different values are demonstrated in table 1. After that the other bytes are stored, with the high bit switching between one and zero followed by the remaining seven bits. [5]

To send a pre-key messages a new chain key and with that a new messages key is derived from the old chain key. The message contains: the public part of the senders identity key, of the ratchet key and of both single-use keys, also the current chain index and of course the message, which is encoded with the message key. The sender keeps sending these pre-key messages, till the addressee responds. [5]

When a participant receives a pre-key message he builds his root and chain key from the identity and single-use keys. The current state of the chain key can be replicated because of the received chain index. With that information he also is able to get the message key and decrypt the received message. [5]

4.3. Main Algorithm

From now on we describe how the algorithm works after the initialisation. At the beginning it is important to know, that the chain keys with an even number are used to encrypt messages from the first participant and the odd ones are used for the second participant. To send a message the sender will check, if a fitting chain key exists, or else he will create a new ratchet key. With that ratchet key a new chain and root key are generated. With the current chain key a message key is build and the message is send. [5]

A normal message consists of a Version Byte, Payload Bytes and Message Authentification Code (MAC) Bytes. The Version Byte is 'x\03' and the Payload Bytes are encoded like the pre-key messages. For the normal messages other tags are valid. These are shown in table 2. The information carried in these Bytes is: the chain index, to find the fitting message key, the public ratchet key and the encrypted real message. The MAC Bytes are part of the MAC, the length is provided by the encryption algorithm. [5]

When receiving a message, the addressee first checks if the ratchet key he receives is the same as his. If not he computes the next ratchet key and with that a new chain and root key. Also he checks if the chain indexes are the same, then he builds a new message key from the chain key, else he takes an old message key, that fits the index and was saved before. With that message key he can now decrypt the received message. [5]

4.4. Differences between the Double-Ratchet Algorithm and Olm

The biggest difference between the Olm and the double ratchet algorithm is that Olm has no sending and receiving chain, but just one in which the index decides about the sender. The Diffie-Hellman key is called ratchet byte in the Olm algorithm.

5. Megolm

In the following part we describe the Megolm protocol, which can protect the communication of many

doi: 10.2313/NET-2019-10-1_05

TABLE 3: Megolm Message Tags

Name	Tag	Type
Message Index	0x08	Integer
Cipher-Text	0x12	String

recipients in a conversation. Every member of the group has an outbound session, with a ratchet chain and a key pair. The key pair is used to authenticate him, so everyone in the group knows, who is sending and who's receiving the message. With the ratchet chain, new message keys are generated, so the safety is guaranteed. If a member wants to share his current ratchet key and his public key, he does this with a peer-to-peer connection to another member. This connection is encrypted with a safe algorithm. For example Olm can be used. [6]

For a safe storage of the server history, like it is provided by Matrix, the servers only save the encrypted messages. The users can only read these from the point when they joined the group, because all used message keys can be built from the first ratchet key they got. [6]

5.1. Initialisation

Every session of each member of a group has a counter, a key pair and a ratchet with four different values. There can be many session in a conversation. The public key helps to authenticate the different sessions. To initialise such a session, the counter is set null and a random value is assigned to the other values. To add new users to this session the session data is shared over a safe peer-to-peer communication, which can be Olm. [6]

The format to share that information consists of exactly 229 Bytes. At the beginning stands the Version Byte with the value 'x\02' followed by the four different ratchet 32-Bit Integers and the public key. It ends with a 64-Bit Signature, showing who sent the data. The receiver of this data checks the signature and saves the other values. [6]

5.2. Main Algorithm

The message key in Megolm is derived from the ratchet. The number of steps that were performed on the ratchet, plus the encrypted message, is sent to the other servers. These messages have a certain format, that has a very similar format to the message from Olm. First the Version Byte 'x\03' is sent, then the encrypted Payload bytes and the MAC Bytes like in Olm. The tags for the Megolm algorithm are described in table 3. The only difference is the signature Byte that is sent at the end, to authenticate the sender of this message. Because the messages are encrypted this good, they can be sent over insecure channels. [6]

Every message should be encrypted with another message key, so after sending, a new key is created. To do that four different hash functions are needed. The ratchet algorithm takes the four different values and changes them after a certain number of iterations. The message key is built from a hash of the combination of the four values. [6]

The value of the ratchet and the counter are stored in the session. The earliest value of the ratchet can be saved to guarantee backward compatibility. [6]

6. Conclusion and future work

Finally it can seen that the Matrix cryptography is a good way to protect our messages. The algorithms are already used by Riot and WeeChat [2]. And also common applications like WhatsApp are using techniques like the end-to-end encryption, which is also provided by Matrix.

Also the idea of connecting all communication platforms, can help many people and make communication less complicated. Matrix makes a standard and save communication possible. To help us protecting our messages more people should use the system. A commercial for developers or for everyone could help to make this network common. At first the development should be finished, to avoid mistakes, which are not identified till yet and could be a huge security lack.

But not everything is perfect, in the Megolm algorithm were found some lacks in the protection of messages. The developing team is already working to fix them [6]

References

[1] "Facebook-Hacker klauten hochsensible Daten," https://www.welt.de/wirtschaft/article182033314/Facebook-Hacker-klauten-sehr-private-Daten-von-Millionen-Nutzern.html.

[2] https://matrix.org/blog/home/, [Online; accessed 07-April-2019].

[3] "Frequently Asked Questions," https://matrix.org/docs/guides/faq, [Online; accessed 07-April-2019].

[4] T. Perrin and M. Marlinspike, "The Double Ratchet Algorithm," https://signal.org/docs/specifications/doubleratchet/, [Online; accessed 07-April-2019].

[5] "Olm: A Cryptographic Ratchet," https://git.matrix.org/git/olm/about/docs/olm.rst, [Online; accessed 07-April-2019].

[6] "Megolm group ratchet," https://git.matrix.org/git/olm/about/docs/megolm.rst, [Online; accessed 07-April-2019].

doi: 10.2313/NET-2019-10-1_05

How Good Is QUIC Actually?

Manuel Burghard, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: burghard@in.tum.de, jaeger@net.in.tum.de

***Abstract*—The Internet Engineering Task Force (IETF) is currently finalizing the standardization of the QUIC core transport protocol and HTTP/3 (H3) with a target date of July 2019 for the final specification. The working group has defined five key goals that should be improved on: connection establishment and transport latency, head-of-line blocking, secure transport, allowing future evolution, as well as multipath and forward error correction extensions.**

In this paper, we explain the problems QUIC tries to solve, discuss QUIC's proposed solutions, and highlight their strengths and weaknesses.

***Index Terms*—networks, multi-layer transport protocol, latency reduction, performance**

1. Introduction

QUIC was initially designed by Roskind [1] at Google to improve the performance of the SPDY protocol running on top of the Transmission Control Protocol (TCP) and Transport Layer Security (TLS). SPDY was the conceptional predecessor of HTTP/2 (H2) which was standardized in 2015 [2]. QUIC merged key functionalities of the transport, security, and application layer into a new protocol based on the User Datagram Protocol (UDP) with the goal of reducing latency and improving performance [1]. In 2015, a first draft for standardization by the Internet Engineering Task Force (IETF) was submitted by Iyengar and Swett [3] and in 2016 a working group was formed [4]. To distinguish between the IETF's and Google's version of QUIC we refer to Google QUIC as gQUIC.

Langely et al. [5] state that Google used its control of the Chrome browser and its web services to create a large scale test and evaluation environment for the development of gQUIC. Initially, a small number of users were randomly selected to test gQUIC in Chrome and the amount was gradually increased until 2017 when only a small control group using TCP and TLS was left. The control over client and server software in combination with a large user base allowed fast and regression free iterations of the protocol [5].

IETF QUIC is a stream-multiplexing UDP-based protocol which always encrypts its payload using TLS 1.3. In contrast to Google's gQUIC implementation the IETF working group decided to separate gQUIC's transport protocol and the adopted H2 application protocol to allow other application level protocols [6]. The adapted version of H2 for QUIC is currently being standardized, too, and will be named HTTP/3 (H3) [7].

The QUIC working group defined five key goals QUIC should deliver [6]:

- Secure the transported payload using TLS 1.3.
- Enable deployment without requiring changes to network equipment along the path.
- Multiplexing without head-of-line blocking, like introduced by H2.
- Minimize the connection establishment and transport latency.
- Enable extensions for forward error correction and multipath connections.

This paper focuses on the IETF version of QUIC in its discussions, but references observations of real world data usage by gQUIC because it is the only large scale deployment as of June 2019.

The remaining parts of this paper are structured in the following way. In Section 2 an overview of related work is provided. Section 3 focuses on strengths and weaknesses of QUIC's key goals. For each goal, we will discuss the problems that should be solved, the solutions proposed by QUIC, and their strengths and weaknesses. Section 4 discusses the overall performance of QUIC in comparison to TCP. Section 5 concludes this paper and discusses future work.

2. Related Work

A short overview of QUIC is presented by Yosofie [8] and a more elaborate introduction of QUIC and H3 is given by Stenberg [9]. The full specification is currently available in a series of IETF drafts [7], [10]–[12]. A comparison between QUIC and Stream Control Transmission Protocol (SCTP) is presented by Joseph et al. [13].

Cook et al. evaluated the performance of QUIC in comparison to H2 based on the page load time [14]. Kakhki et al. [15] present the results of their performance comparison of TCP and QUIC using different environments and by running the tests with different versions of the QUIC implementation of the Chromium project to compare the development over time. QUIC was originally intended to be implemented in user space, but Wang et al. implemented QUIC in the Linux kernel and compared its performance to TCP [16].

In 2013 Multipath TCP (MPTCP) extensions were standardized by Ford et al. [17] which define the usage of multiple (disjoint) paths through a network to provide TCP's bi-directional stream. Cheng et al. defined TCP Fast Open (TFO) [18] which may reduce the connection establishment by one round-trip time (RTT) for TCP, but

doi: 10.2313/NET-2019-10-1_06

also introduces a number of new security implications, like resource exhaustion or amplified reflection attacks.

3. Strengths and Weaknesses

In this Section, we evaluate the strengths and weaknesses of the key goals set out by the QUIC working group. Each subsection focuses on one of the goals, explaining the existing problem and QUIC's proposed solution.

3.1. Secure Transport

The leaks and uncoverings by journalists of the last decade have proven those correct who always warned about the interest of governments and private corporations in private user data [19], [20]. Nowadays, it is broadly acknowledged that user data should be encrypted when it is sent over the Internet to ensure a minimum level of integrity, confidentiality and privacy. The HTTP protocol up to version two relies on TLS on top of TCP for its (optional) encryption. When opening a connection to a server, first the three-way handshake of TCP [21] is performed followed by the TLS handshake [22] to establish a secure bi-directional connection between a client and a server. During the standardization of H2, the working group discussed making always-on encryption part of the protocol, but was unable to find consensus on this topic [2], [23].

Google's first specification of QUIC from 2012 [1] already required always-on encryption for the payload and some header fields. More information about why some header fields are encrypted will be provided in Section 3.2. Google's initial design for QUIC predated TLS 1.3 and therefore Langley et al. came up with their own encryption [24]. The QUIC working group later decided to use TLS 1.3 for encryption [10], [12].

Always encrypting the payload benefits any potential end user of the protocol by keeping their communication private. It also forces companies or other organizations who wish to use QUIC to provide secure transport, which should lead to a broader adoption of encrypted communication on the Internet. On the other hand, there are companies like banks which have to meet certain regulatory or compliance requirements which effectively prevent the usage of TLS 1.3 due to being too secure and constrains these companies to standards like Enterprise Transport Security (ETS) [25], which was recently assigned a CVE number for its lack of per-session forward secrecy [26].

The usage of TLS 1.3 comes at a cost, too: QUIC is vulnerable to application layer replay attacks when using 0-RTT, similar to TLS 1.3, and application layer protocols must include mitigations [12].

3.2. Enabling Future Changes to QUIC

Iterating on a protocol like TCP or trying to introduce a new protocol like SCTP often requires long adoption time because the network equipment along the paths must support and understand the traffic. Especially router and firewall vendors are known to make certain assumptions about the protocols they are supporting, like dropping TCP packets which contain unknown or new TCP options [9]. Any unexpected change may lead to packets being flagged as illegitimate which results in rejecting or dropping said packets. For example, adoption of SCTP on top of IP is still not widespread today [13]. This stiffness of the existing Internet infrastructure is referred to as *ossification* [5], [9].

In addition to that, protocols like TCP or SCTP are usually implemented in the kernel and require the commitment of operating system vendors, changes to that implementation are bound to operating system updates, and users must install those updates until an application can rely on a new protocol. As of June 2019 SCTP is still not supported by Microsoft Windows or Apple's operating systems, but third party implementations exist [27].

QUIC tries to solve the problem of adoption and allowing for future changes: First of all, it is built on top of UDP. Existing network equipment already supports UDP and does not require support for a completely new transport layer protocol. Second, QUIC can be fully implemented in the user space which makes it independent of any support in an operating system kernel. A downside of this approach is a potential performance penalty induced by system calls and context switches, but a user space implementation decouples development from the operating system release schedule and allows faster iteration and easier deployment to legacy systems. Applications can include a QUIC library and can distribute newer versions independent of the operating system they are running on, like Google is doing with Google Chrome. The current adoption rate also speaks for QUIC's approach: Langley et al. [5] estimated the amount of QUIC traffic on the Internet at 7% in 2017, about five years after the introduction. In addition to that, QUIC also includes version negotiation in the protocol as part of opening a connection [10], thereby enabling the introduction of new QUIC versions while allowing clients and servers to agree on a version supported by both sides.

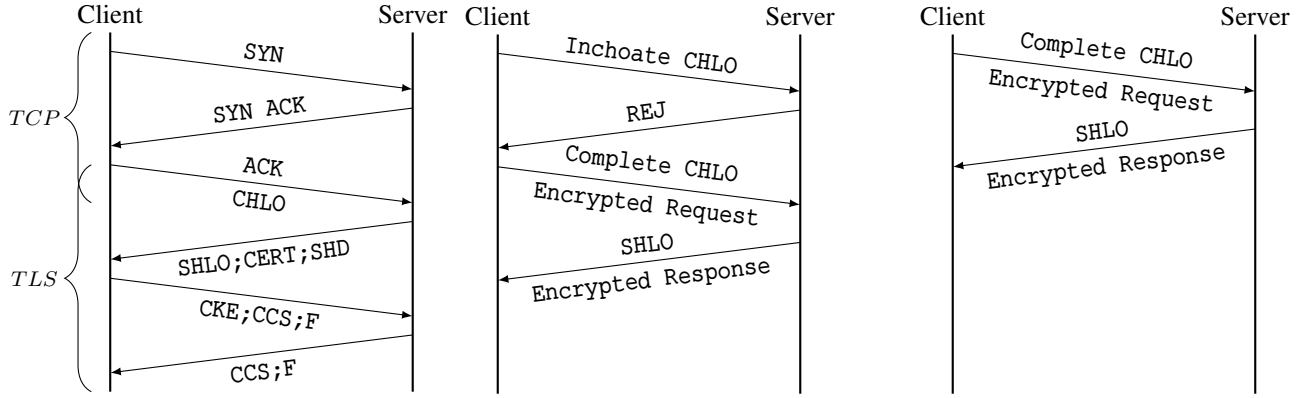
To prevent ossification, QUIC tries to encrypt as much data as possible, including signaling information [10], to hide it from network equipment and prevent vendors of said equipment from making assumptions that will interfere or prevent future changes to the protocol.

The strengths of the solutions provided are obvious: The adoption rate is already significant and precautions have been taken to simplify changes to the protocol in the future. Nevertheless, 4.7% of video playback traffic is having problems due to network equipment blocking, rate limiting, or otherwise limiting UDP traffic [5].

3.3. Head-of-Line Blocking

Head-of-line (HOL) blocking describes the situation of sequential packets or requests being held up by the first item in a serial queue. HOL blocking can appear on different network layers like TCP when a packet is dropped/delayed or in HTTP/1.1 (H1.1) when all open TCP connections to a server are already transferring requests and additional request are forced to wait until another request has finished. With the introduction of H2 and its multiplexed streams the number of connections opened by e.g. a browser could be reduced to one compared to the up to six from H1.1 while also allowing multiple requests

doi: 10.2313/NET-2019-10-1_06



(a) TCP and TLS handshake (simplified).

(b) QUIC 1-RTT connection establishment [5]

(c) QUIC 0-RTT connection establishment [5]

Figure 1: Comparison of TCP and TLS, QUIC 1-RTT, and QUIC 0-RTT handshakes.
Abbreviations used in the diagrams: ClientHello (CHLO), ServerHello (SHLO), Certificate (CERT), ServerHelloDone (SHD), ClientKeyExchange (CKE), ChangeCipherSpec (CCS), Finished (F), Reject (REJ)

being performed at the same time. Therefore, H2 may be less likely to suffer from application layer HOL blocking in comparison to H1.1, but is more prone to HOL blocking on the TCP layer [9], [28].

Similar to H2, QUIC supports multiple streams over a single connection, but loss detection and recovery are part of QUIC itself and not of an underlying protocol layer like TCP. UDP is connectionless and does not provide any loss detection or recovery at all. In case of lost packets, recovery only impacts the streams whose frames were part of the lost packets. Other streams are not affected by the recovery and hence not blocked. Retransmission is also different when compared to TCP. Instead of retransmitting a lost packet, QUIC checks for every lost stream frame whether the contained data is still needed. If that is the case, the stream frames and packets are then retransmitted. If a stream is reset in the meantime, the lost frames for this stream are not retransmitted [10], [11].

Eliminating HOL blocking leads to better performance of QUIC in lossy environments. The tests performed by Kakhki et al. [15] support this claim for desktop and mobile environments, although the gains are inferior in the mobile environment due to slower packet consumption of QUIC's userspace implementation. The downside is that loss detection and recovery were reimplemented on top of UDP although TCP implementations already exist and are well tested.

3.4. Connection Establishment and Transport Latency

The main goal set out initially by Google was to reduce latency, especially for connection establishment which usually includes some form of handshake, like the well known TCP three-way handshake [1]. For an H2 connection to a server, first a TCP connection is opened, followed by a TLS handshake if an encrypted connection is desired. The full handshake flow for TCP and TLS is shown in Figure 1a and results in a minimum of three round trips until a connection is established and application data can be transferred. In an high delay environment, the handshake latency can highly influence the perceived performance of the network connection and thereby degrade the user experience.

QUIC improves on connection establishment by combining the transport and cryptographic handshake [10]. This results in a 1-RTT handshake which means only one round-trip is needed until application data can be transfered using the newly established connection. Furthermore, QUIC also supports 0-RTT handshakes, which use cryptographic information from a previous connection for even faster connection establishment allowing data transfer to start with the initial message from a client. Both handshake variants can be seen in Figure 1b and 1c. Data released by Google on handshake latencies shows that TCP and QUIC handshake latencies are growing linearly with growing RTTs but QUIC's slope is lower due to 0-RTT [5]. Even when considering just the 1-RTT connection establishments, QUIC only takes about half the time of TCP. The benefit of supporting 0-RTT handshakes was further shown by [15] who compared QUIC with and without enabled 0-RTT handshakes and found significantly improved performance for small transfers up to 10 kB. There was an attempt to reduce the handshake latency for TCP with TFO in [18], but according to Paasch it suffers from ossification and the success rate is at about 80% [29]. Google's servers support TFO but the influence of TFO for their handshake comparison is not shown or highlighted [5].

TLS 1.3 support for 1-RTT and 0-RTT handshakes is not limited to QUIC and can can be used in the underlying connection of an H2 stack to further improve the connection establishment latency in the classic HTTP stack. This could diminish the performance gains of QUIC in comparison to TCP and TLS.

3.5. Multipath and Forward Error Correction

QUIC's multipath extension is not part of the July 2019 milestone but adoption is scheduled for December 2019 and the handover to the Internet Engineering Steering Group (IESG) is scheduled for May 2020 [6]. De Coninck et al. present an experimental implementation of QUIC multipath with the goals of resilience to connection

	Search latency	Video latency	Video rebuffer rate
Desktop	8.0	8.0	18.0
Mobile	3.6	5.3	15.3

TABLE 1: Mean percent reduction of the search latency, video latency, and video rebuffer rate observed by [5].

failures and the combination of available resources, and compare its performance to MPTCP [30]. They are also the authors of the current IETF draft for QUIC multipath [31]. Forward Error Correction (FEC) is out of scope of the first standardization [6]. Google supported FEC in gQUIC but removed it in 2016 due to the unconvincing results and the increase in code complexity [5]. Due to these reasons we do not further discuss multipath and FEC.

4. Overall Performance

In this section we want to compare different reports about QUIC's performance. We focus on three papers starting with Langley et al. [5].

In 2017 Google presented the results and measurements of their production deployment of QUIC for all Google services with probably the largest sample size [5]. Their observations of handshake latencies were already summarized in Subsection 3.4. Google evaluated the performance of QUIC in comparison to TCP and TLS using the search latency, video latency, and video rebuffer rate as metrics. We only summarize the mean percent reduction in Table 1 and refer to [5] for more details. Clearly visible in the data provided are the inferior gains of QUIC in the mobile environment. This is explained by a lower 0-RTT handshake rate due to mobile devices changing IP addresses when switching networks and, thereby invalidating the cached cryptographic information for the handshake. Another reason for a lower 0-RTT handshake rate is hitting different servers causing a 0-RTT handshake to fail, too. The results for video latency are additionally influenced by the YouTube app which performs handshakes in the background to improve latency. The video rebuffer rate measures the time spent on rebuffering data during video playback and is not directly dependent on the handshake latency. Instead, it depends on loss recovery and overall throughput of the established connection. According to Langley et al. QUIC performs best in high delay, low bandwidth, and lossy networks. This claim is further supported by comparing QUIC's performance when used in India to the performance in South Korea with both countries being on opposite ends of Internet quality scale with regard to delay, loss, and throughput. During the tests, they also noticed higher CPU usage for QUIC. Even after optimizations QUIC doubled TCP and TLS CPU usage. Stenberg [9] mentions the slowness and higher CPU consumption of QUIC, too, and explains it with the lack of hardware acceleration and lack of optimized UDP stacks.

Cook et al. [14] evaluated QUIC's performance based on the Page Load Time (PLT) of websites in different scenarios with regard to delay, loss, network type (cellular vs ADSL), and network load in comparison to H1.1 and H2 on top of TCP and TLS. The test environment consisted of copies of websites hosted on virtual machines and their real world counterparts. Their tests showed that QUIC performs better under delay, in lossy networks, and when connected to a cellular network. These results match with the observations of Langley et al. described above. In addition to the PLT evaluation, the influence of the distribution of a website was investigated showing that QUIC performs better if the number of servers hosting website resources is low.

Another evaluation of QUIC's performance was done by Kakhki et al. [15] using Chrome and the corresponding server component [32]. The scenarios compared QUIC to TCP based on PLT in a desktop and mobile environments, video streaming performance, the fairness of QUIC with regard to sharing of bottleneck bandwidth, and the impact of in-network proxying when using QUIC. The performance characteristics of QUIC for the PLT match with the results we presented before: QUIC performs better than TCP for small objects and connections with loss, the gains on the desktop are higher than those on mobile devices. Interesting findings include QUIC's poor performance when packet reordering is required, resulting in TCP outperforming QUIC. This is explained by the threshold for negative-acknowledgments being based on a fixed number in QUIC causing it to start retransmission whereas TCP benefits from dynamically adapting the threshold. During the setup of the testbed, the authors noticed that the QUIC server's default parameters result in worse performance when compared to Google's production QUIC servers and therefore tweaked the parameters until the performance matched that of Google's servers. An unfairness of QUIC was observed when comparing the consumption of bottleneck bandwidth: QUIC vs QUIC results in equal shares, but QUIC vs TCP results in an unfair imbalance towards QUIC although both used the same congestion control algorithm.

5. Conclusion and Future Work

One interesting area of future work is to compare QUIC's multipath capabilities to those of MPTCP when the associated milestone is done by the IETF working group. MPTCP is already implemented in major operating systems like Linux [33]. Additional future work may focus on performance comparisons of H2 over TCP and TLS, especially how 1-RTT and 0-RTT handshakes of TLS 1.3 influence the results.

The design of QUIC is the logical consequence of combining the benefits of recent network protocols and continuing good ideas a step forward. Always-on encryption is not just for the benefit of the end user's privacy, but prevents network ossification. The possibility of an user space implementation allows deployment independent of any minimal operating system version and rapid iterations. Faster handshakes and the elimination of HOL blocking lead an improved user experience. Overall, the strengths mentioned in this paper outweigh the weaknesses. QUIC may not replace TCP and TLS immediately, but there are areas like high latency and lossy networks or cellular networks where QUIC is well suited to take over.

References

[1] J. Roskind, "Quic: Multiplexed stream transport over udp," https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34, 2012, [Online: accessed 10-June-2019].

[2] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt

[3] J. Iyengar and I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," https://tools.ietf.org/id/draft-tsvwg-quic-protocol-00.txt, 2017, [Online; accessed 04-June-2019].

[4] M. Westerlund and S. Dawkins, https://datatracker.ietf.org/doc/charter-ietf-quic/00-00/, 2016, [Online; accessed 04-June-2019].

[5] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842

[6] Internet Engineering Task Force, "QUIC (quic)," https://datatracker.ietf.org/wg/quic/about/, [Online; accessed 04-June-2019].

[7] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-20

[8] M. Yosofie, "Recent progress on the QUIC protocol," in *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Winter Semester 2018/2019*, ser. Network Architectures and Services (NET), G. Carle, S. Günther, and B. Jaeger, Eds., vol. NET-2019-06-1. Munich, Germany: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich, Jun. 2019, pp. 77–81.

[9] D. Stenberg, "HTTP/3 explained," https://http3-explained.haxx.se/en/, 2018, [Online; accessed 04-June-2019].

[10] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-20

[11] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-recovery-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-20

[12] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-tls-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-20

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

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

[15] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic: An approach for rigorous evaluation of rapidly evolving transport protocols," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, pp. 290–303. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131368

[16] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the quic protocol in linux kernel," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: ACM, 2018, pp. 227–234. [Online]. Available: http://doi.acm.org/10.1145/3242102.3242106

[17] A. Ford, C. Raiciu, M. J. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Jan. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6824.txt

[18] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," RFC 7413, Dec. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7413.txt

[19] G. Greenwald, "NSA collecting phone records of millions of Verizon customers daily," Jun. 2013, [Online; accessed 21-June-2019].

[20] K. Roose, "Facebook Emails Show Its Real Mission: Making Money and Crushing Competition," Dec. 2018, [Online; accessed 21-June-2019].

[21] "Transmission Control Protocol," RFC 793, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt

[22] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[23] "Http/2 frequently asked questions," https://http2.github.io/faq/#does-http2-require-encryption, [Online: accessed 10-June-2019].

[24] A. Langley and W.-T. Chang, "Quic crypto," https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g, 2013, [Online: accessed 10-June-2019].

[25] T. Rutkowski and S. Compans, "ETSI TS 103 523-3," https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf, European Telecommunications Standards Institute, Technical Specification, Oct. 2018, [Online; accessed 16-June-2019].

[26] "CVE-2019-9191," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9191, 2019, [Online; accessed 23-June-2019].

[27] M. Tüxen, "usrsctp," https://github.com/sctplab/usrsctp, [Online: accessed 15-June-2019].

[28] D. Stenberg, "http2 explained," https://daniel.haxx.se/http2/, 2014, [Online; accessed 15-June-2019].

[29] C. Paasch, "Network Support for TCP Fast Open," 2016, presented at NANOG67 in Chicago, IL. [Online]. Available: https://archive.nanog.org/sites/default/files/Paasch_Network_Support.pdf

[30] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: ACM, 2017, pp. 160–166. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143370

[31] Q. D. Coninck and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)," Internet Engineering Task Force, Internet-Draft draft-deconinck-quic-multipath-03, Aug. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-03

[32] "The chromium projects," https://www.chromium.org, [Online; accessed 18-June-2019].

[33] "MultiPath TCP - Linux Kernel implementation," http://multipath-tcp.org/pmwiki.php/Main/HomePage, [Online; accessed 16-June-2019].

Porting ixy.rs to Redox

Simon Ellmann, Paul Emmerich*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ellmann@in.tum.de, emmericp@net.in.tum.de

***Abstract*—Drivers are traditionally written in C and make up a huge part of every operating system. 66% of all code in the Linux kernel is driver code, and the current number of drivers and their complexity are still increasing. However, this complexity comes at a price of decreasing readability of the code and greater vulnerability. Some of the problems related to drivers, especially concerning their safety and security, can be mitigated by running drivers in user space. This is especially beneficial on modern microkernel architectures where the operating system can seamlessly interact with user space drivers. A good example for such an operating system is Redox, a Unix-like OS written in Rust.**

In this paper, we implement the first 10 Gbit/s user space network driver for Redox by porting an existing Linux implementation, ixy.rs. We evaluate the driver's overall structure, integration into the operating system and performance and compare it to the original implementation and Redox's other network drivers. We show that our driver is many times faster than Redox's other drivers although it uses less unsafe code. Our code is available as free and open source under the AGPL-3.0 license at https://github.com/ackxolotl/ixgbed.

***Index Terms*—Rust, User Space Driver, Redox, Performance, Microkernel**

1. Introduction

Up until the 1940s, computers could only perform series of single tasks. Today well known features like scheduling, memory management and multitasking did not exist in operating systems – as far as programs were run in operating systems at all – until the 1960s when hardware abstraction became prevalent. In 1969, the development of Unix started, an operating system containing most of its functionality inside a big kernel, thus forming one of the first monolithic kernel architectures. The development of Unix set a variety of standards for today's operating systems. Popular operating systems like Windows, macOS and Linux-based ones like Debian or Ubuntu are still built on monolithic or hybrid kernel architectures.

Although monolithic kernels are very common and might be easier to implement, they are considered obsolete by researchers since the 1990s (see the Tanenbaum–Torvalds debate). This is due to various flaws in their architecture: Programming mistakes in the kernel can take down the whole system or corrupt other processes since every piece of software in the kernel is executed with full privileges, development of new software is tedious because common libraries and debuggers are missing and maintenance of the kernel can be challenging if its complexity is rapidly increasing (e.g. like the Linux kernel).

A more temporary approach to operating system design are microkernels. Unlike monolithic kernels, microkernels try to minimize the amount of software running in kernel space by moving almost all applications to user space. This includes (but is not limited to) the following services running in user space: Drivers, file system and inter-process communication. Keeping the kernel small yields various advantages: Kernel components traditionally written in C can be rewritten in other programming languages, execution of this software can be easily debugged in user space and faults in user space daemons have no impact on the overall system.

So why is all this relevant? In 2019, Cutler et al. evaluated security bugs leading to arbitrary code execution in the Linux kernel [1]. Of the 65 bugs published in the CVE database in 2017 with patches available, 40 were memory bugs that could have been prevented by using a memory-safe language like Go or Rust. Of these 40 memory bugs, 39 were located in device drivers. Since 66% of the code in the Linux kernel is driver code [2], the findings of Cutler et al. reveal that drivers offer a large attack surface and many possibilities for improvement.

Rewriting drivers in memory-safe programming languages can mitigate many safety and security faults. An example for such a driver is ixy.rs [3], a rewrite of the simple user space network driver ixy [4] in Rust for Linux. Unfortunately, Linux is not particularly suitable for user space networking due to its monolithic kernel design: The OS network stack cannot be used by user space drivers and memory allocation for the PCIe device is only possible by using a quirk in Linux. However, there are other operating systems based on microkernels like Redox [5], a Unix-like operating system written in Rust, that implement full network functionality in user space.

In this paper, we try to combine the advantages of a user space network driver and an operating system based on a microkernel, both written in a memory-safe programming language, by porting ixy.rs to Redox. The common denominator of ixy.rs and Redox is Rust [6], a novel programming language illustrated in Section 2. Section 3 introduces Redox, while the following Section is about ixy. In Section 5 we evaluate ixy on Redox, Section 6 presents related work to the inclined reader. We summarize our results in Section 7 and present opportunities for future work in the area of ixy.rs on Redox.

The main contribution of this paper is the first 10 Gbit/s network driver on Redox [7].

doi: 10.2313/NET-2019-10-1_07

2. Rust

Rust is a relatively new systems programming language focusing on memory- and thread-safety. Its first stable version was released in May 2015. While Rust provides zero-cost abstractions like C++ and is also syntactically similar, its main selling point is memory safety due to its novel ownership system [6].

2.1. Type System

Rust is statically typed, i.e. the types of all variables and functions are checked at compile time. Functions have to be annotated by programmers explicitly, types of variables can be inferred in most cases by the Rust compiler. The type system provides "traits", i.e. interfaces that can be implemented by multiple types similar to type classes in Haskell, and generic parameters to allow for inheritance and ad hoc polymorphism.

2.2. Memory Management

Rust's core feature is its unique ownership system which enforces Rust's guarantees of memory safety and data-race freedom. While many programming languages make use of garbage collectors, Rust ensures at compile time that memory is allocated, handled and freed correctly. This yields two major advantages compared to garbage collection:

1) Memory handling, especially cleanup of resources, is deterministic.
2) There are no performance issues for real-time applications caused by garbage collection.

Unlike in C or C++, it is not possible in (safe) Rust to build a program leading to undefined behaviour by freeing memory twice, accessing dangling pointers or other operations violating memory safety due to the additional rules that Rust enforces on memory handling. Since Rust verifies memory safety at compile time and not at runtime, there is no size or performance overhead in compiled programs [8].

2.3. Ownership

The ownership system of Rust ensures that every value in Rust has a unique owner and that the scope or lifetime of a value depends on the scope/lifetime of its owner, i.e. if the owner of a value goes out of scope the value is freed (similar to *Resource Acquisition Is Initialization* (RAII) from C++) [6]. Ownership can be transferred between variables, values are either copied or moved in memory depending on whether the value is stored on the stack (and it is thus cheap to copy the value) or it is stored on the heap. Where a value is placed in memory usually depends on whether the size of a value is known at compile time or not. Values can be passed to functions by immutable or mutable references, or by value. As long as there is a reference to a value, the value cannot be moved (to inhibit dangling pointers). There can be multiple immutable references to a value or a single mutable reference. While a mutable reference to a value exists, i.e. the value is borrowed mutably, the value can only be modified through that reference and not through its owner to prevent data races.

2.4. Safe and Unsafe

The ownership system of Rust is very powerful. However, static analysis is quite conservative and still subject to limited decision capabilities. There are valid programs that are rejected by the compiler when the compiler is unable to decide whether the code upholds the required guarantees. This is always the case for programs that

- call foreign functions (e.g. from `libc`),
- dereference raw pointers,
- access and modify mutable static variables or
- call unsafe functions or implement unsafe traits.

These features can be used inside an `unsafe` block. In unsafe code the developer has to ensure that the program obeys the memory guarantees of Rust. Unsafe code in Rust is nothing unusual, e.g. many parts of the Rust standard library make use of unsafe code. Nevertheless, by verifying parameters before and return values after unsafe code, developers ensure that the unsafe operations are actually safe, thus forming safe wrappers around unsafe code.

3. Redox

Redox is an operating system written in Rust. It was published in 2015 by Jeremias Soller, is actively maintained since then and has received over 2,000 contributions by more than 70 developers. Similar to the Rust programming language, Redox focuses on safety, reliability and eventually performance [5]. To achieve these goals, the Redox developers opted for a microkernel architecture similar to MINIX [9]. Redox's developers try to "generalize various concepts from other systems, to get one unified design" [10], namely concepts from Plan 9 [11], Linux and BSD.

3.1. Everything is a URL

Redox generalizes Unix's "everything is a file" with its concept of "everything is a URL" [10], i.e. URLs, schemes and resources are used as communication primitives between applications. URLs model segregated virtual file systems that can be arbitrarily structured. They consist of two parts separated by a colon, the scheme (e.g. `file`) and a reference part (e.g. `/usr/bin/ping`). URLs identify resources like genuine files in the filesystem, websites, hardware devices and other primitives. Schemes are created by the kernel or user space daemons. They are registered by opening the name of the scheme in the root scheme (which defaults to empty), i.e. to create the `file` scheme a process has to open `:file` with the `CREATE` flag. Accesses to a URL are processed by the scheme-registrar which returns a handle to the requested resource, e.g. a file descriptor. Resources behave either file- or socket-like, i.e. reads and writes are buffer- or stream-oriented.

doi: 10.2313/NET-2019-10-1_07

3.2. Drivers in Redox

As is to be expected with a microkernel architecture, drivers in Redox operate as user space daemons. PCI drivers are launched on boot by Redox's PCI driver manager, `pcid` which in turn is launched by Redox's init process. `pcid` parses a configuration file associating PCI device vendors and classes with Redox's drivers and their command line parameters, i.e. name of the device, location of Base Address Registers (BARs), etc. Drivers have to implement various functions like `open`, `read`, `write`, `close`, etc. to communicate with other applications via Redox's URL API. Network drivers have to register the `network` scheme.

Communication between other user space programs and the driver is handled via socket-like resources.

4. ixy

ixy is a light-weight user space network driver written for educational purposes [4]. It is a custom reimplementation of Intel's ixgbe driver for 10 Gbit NICs. ixy's architecture is inspired by DPDK [12] and Snabb [13]. ixy does not rely on a kernel module (like Snabb) and features memory management of DMA buffers with custom pools, polling instead of an interrupt-driven design and an API that supports batch operations for receiving and transmitting packets (like DPDK). ixy was originally written in C by P. Emmerich et al. in 2017 but has been ported to more than ten other programming languages including Go, Haskell, Python and Rust (also known as ixy.rs).

We will describe the architectural design and the implementation of ixy on Redox in the following subsections. To understand how ixy and similar drivers work it is necessary to understand how the driver and the device communicate with each other. There are two communication channels for PCIe devices: The driver can access the device's configuration registers (BARs) to control the device and the device can access main memory via direct memory access (DMA) to read and write packet data and packet status information [4].

4.1. Memory Management

While ixy makes use of custom memory pools which are a reasonable choice for user space drivers on Linux due to missing tools for allocating and managing DMA memory in user space and to gain high performance, ixy.rs on Redox does not use custom memory pools for two reasons:

1) Redox provides an API for handling DMA memory in user space.
2) Performance of the driver is restricted due to its interrupt-driven design and the fact that packets cannot be processed in batches. These performance barriers cannot be mitigated by custom memory pools.

ixy.rs on Redox allocates all DMA memory via Redox's syscall API. Accesses to the device's registers (BARs) happen via memory mapped IO: The device is mapped into the memory space of the driver, read and write operations on the registers lead to data transfer on the PCIe bus.

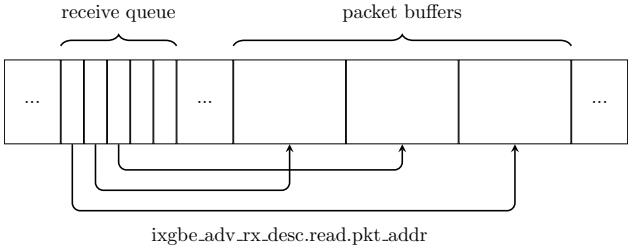


Figure 1: DMA memory containing receive queue with descriptors pointing to packet buffers.

4.2. Receiving Packets

NICs provide multiple ring buffers to receive and transmit packets. Incoming traffic can be split with filters if multiple queues are configured [4]. For the sake of simplicity, ixy on Redox uses only one receive and one transmit queue. Receive and transmit queues work in a similar way: Every queue is a ring buffer filled with descriptors that point to the memory address of the corresponding packet and contain status information about the packet, i.e. size of a received packet or an indicator whether a packet queued for transmittion has been sent out by the NIC yet. The structure of a receive queue is illustrated in Figure 1. Transmit and receive queues are managed by the driver and the device on a rotating basis. The device indicates its current position in the ring via the head pointer, the driver via the tail pointer [4]. Both pointers can be accessed through the BARs of the device.

Before receiving packets, the driver has to initialize the descriptors in the receive queue with physical addresses. For every incoming packet, the NIC writes the packet's data to the memory address given in the descriptor, updates the descriptor and increases the queue's head pointer.

Whether new packets have arrived can be checked by reading the head pointer. This is what the e1000 driver and rtl8186 driver of Redox do [14]. Since accessing the head pointer of the queue incurs a PCIe round trip, a better way to check for new packets is to read the descriptor status field from DMA memory that is effectively kept in the CPU cache [4].

With its custom memory pools, ixy on Linux maintains a stack of free buffers and tracks which buffers are currently in use by the device and the driver. When reading a packet, the corresponding buffer is passed to the user application and the physical address of the descriptor is updated to an unusued buffer from the free stack. Unfortunately, ixy on Redox cannot pass its memory to other user space applications and thus has to copy all received data. However, this obviates the need for a free stack and simplifies the driver: All buffers can be immidiately reused after copying the packet data, no addresses in the descriptors have to be changed.

4.3. Transmitting Packets

Transmitting packets works similarly to receiving packets but is more complicated as packets are sent

doi: 10.2313/NET-2019-10-1_07

asynchronously for performance reasons. The transmit functions consists of two parts: verifying if packets from previous calls have been sent out and putting the current packet in the transmit queue. The first part is usually called cleaning and is executed for the first time when every descriptor of the transmit queue has been used once, i.e. the transmit functions keeps track of used descriptors and runs its cleaning part when the counter of free descriptors equals zero. Cleaning works as follows: The status flag of the descriptor after the last cleaned descriptor is checked. If the descriptor is done, i.e. the packet has been sent out, the next descriptor is checked and the clean index and the counter of free descriptors are increased by one. If the descriptor is not done yet or the whole queue was cleaned, cleaning is finished and the packet to be sent is put into the transmit queue by copying the packet's data to the descriptor's buffer, updating the descriptor (e.g. setting the packet size) and increasing the tail pointer of the transmit queue.

4.4. Interrupts

The official ixy driver works in poll-mode only [4] and does not support interrupts yet. This is not a technical restriction but a performance decision since Linux offers full interrupt support in user space. However, T. Zwickl has implemented interrupt-handling for the original ixy driver [15]. Based on his work we have added support for MSI-X interrupts to ixy on Redox as well. Unfortunately, Redox does not feature MSI-X interrupts yet (which will hopefully change in the future).

4.5. Offloading Features

Although NICs of the ixgbe family support various offloading features, and frameworks like DPDK make use of these features, ixy only enables CRC checksum offloading to keep the driver's complexity low [4].

5. Evaluation

Redox is still in an experimental development state and subject to major changes. No stable version has been released yet. It is possible to boot Redox on real hardware but this requires a hard disk with no partition table [10]. For development purposes it is preferable to run Redox in a virtual machine, e.g. in QEMU which we used to conduct some performance measurements. The results of the following subsections confirm that Redox and its components (e.g. its other network drivers) are still in a very premature development state. However, our implementation is a valuable contribution to Redox from both the performance and the operational safety point of view.

5.1. Performance

We wrote a simple packet forwarder and packet generator application called `rheinfall`[1] to assess the transmit capabilities of our driver. All measurements were performed on commit `bccd1ca` of our implementation.

1. https://github.com/ackxolotl/rheinfall

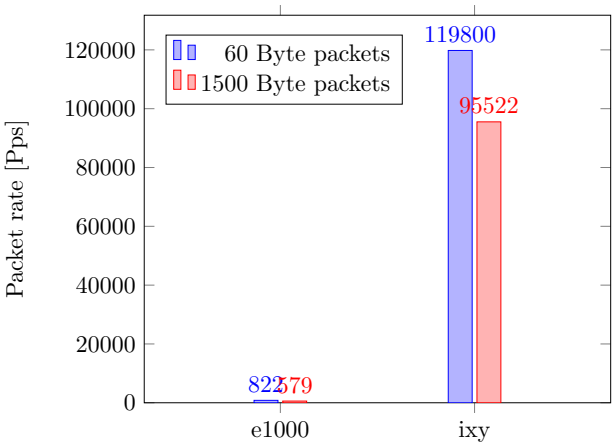


Figure 2: Packet transmit rates measured on AMD Ryzen 7 1800X with Intel 82574L (e1000) and Intel 10G X550T (ixy).

We run `rheinfall` on an AMD Ryzen 7 1800X at 3.6 GHz with Redox 0.5.0 in QEMU with KVM using an Intel X550T NIC via PCIe passthrough through the AMD IOMMU. `rheinfall` bypasses Redox's network stack `smoltcp` by accessing the driver directly, i.e. receiving and sending raw ethernet frames from/to `network`, the scheme registered by the driver. Our measurements show that ixy on Redox can transmit up to 120,000 packets per second with a size of 60 Bytes or up to 100,000 packets per second with a size of 1,500 Bytes which is equivalent to a transmit rate of about 1.1 Gbit/s.

Reasons for this rather poor performance are probably OS-dependent: a not extensively optimized kernel, sluggish interprocess-communication, many context switches for large queues of packets and the fact that data has to be copied multiple times from the application to the driver to the NIC and the other way around. Nevertheless, compared to Redox's two other network drivers, the e1000 and rtl8168 driver, this is a very reasonable packet rate. Figure 2 shows the transmit capabilities of Redox's two Intel network drivers, the e1000 and our implementation. The transmit rates of Redox's e1000 and rtl8168 driver are many times smaller than ixy's (around 90 to 150 [!] packets per second on emulated hardware or up to 1,000 packets on real hardware) due to the fact that – unlike in ixy – their transmit functions block until a packet has been sent out. This simplifies the transmit function but also has a massive impact on performance.

5.2. Safety

Driver	NIC Speed	Code [Lines]	Unsafe [Lines]	% Unsafe
Our implementation	10 Gbit/s	901	68	7.5%
e1000	1 Gbit/s	421	117	27.7%
rtl8168	1 Gbit/s	399	114	28.6%

TABLE 1: Unsafe code in different Redox drivers, counted with `cloc`.

Another point to note is the different amount of unsafe code in our implementation and the other two network drivers in Redox shown in table 1. Unlike the e1000 and

doi: 10.2313/NET-2019-10-1_07

rtl8168 driver, our implementation provides safe functions to read and write the device's registers by asserting that the memory address of a register is indeed inside of the mapped memory region. This optimization alone leads to a few hundred lines less unsafe code.

6. Related Work

As early as 1993, researchers proposed to move network software traditionally implemented in kernel space to user space. Exemplary for these efforts is the work of Chandramohan A. Thekkath, Thu D. Nguyen, et al. [16], in which they suggested to rewrite transport protocols as user-level libraries. Their work already includes a multitude of observations on different kernel designs and the resulting advantages and disadvantages for software. They claim that it is possible to implement protocols in a highly performant and secure way in user space.

Another scientific paper is "The Case for Writing Network Drivers in High-Level Programming Languages", in which P. Emmerich, S. Ellmann et al. present a network driver written in various high-level programming languages [2]. They propose to rewrite drivers instead of the whole operating system in memory-safe languages.

7. Conclusion and Future Work

Many bugs in current operating systems are located in driver code. By moving driver code to user space and using high-level languages (preferably memory-safe ones like Go and Rust), many safety and security related bugs can be mitigated. However, this requires a different kernel design. Modern microkernel architectures like the Redox kernel provide such a design. They form a safe alternative to the deprecated monolithic kernel design of the Linux kernel. In line with the proposal of P. Emmerich, S. Ellmann et al. to port drivers to high-level languages and user space, we present the first 10 Gbit/s user space network driver on Redox written in Rust. Our evaluation shows that the driver is a great contribution to Redox. However, there is still a huge potential and need for optimization on part of the operating system.

Future work on the driver might include implementing the use of multiple receive and transmit queues, further reducing the amount of unsafe code or enabling more hardware offloading features like VLAN tag offloading. Furthermore, depending on the development status of Redox, more detailed performance measurements could be performed, possibly using a high-speed packet generator like MoonGen [17].

References

[1] C. Cutler, M. F. Kaashoek, and R. T. Morris, "The benefits and costs of writing a posix kernel in a high-level language," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 89–105.

[2] P. Emmerich, S. Ellmann, F. Bonk, A. Egger, T. Günzel, A. Obada, M. Stadlmeier, S. Voit, S. Huber, and G. Carle, "The Case for Writing Network Drivers in High-Level Programming Languages," 2019.

[3] S. Ellmann, "Writing Network Drivers in Rust," 2018.

[4] P. Emmerich, M. Pudelko, S. Bauer, and G. Carle, "User Space Network Drivers," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2018, pp. 91–93.

[5] "Redox," https://www.redox-os.org/, accessed: 2019-06-23.

[6] S. Klabnik and C. Nichols, *The Rust Programming Language*. No Starch Press, 2018.

[7] S. Ellmann, "Ixgbe user space driver for Redox," https://github.com/ackxolotl/ixgbed, 2019, accessed: 2019-06-24.

[8] J. Blandy and J. Orendorff, *Programming Rust: Fast, Safe Systems Development*. " O'Reilly Media, Inc.", 2017.

[9] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Minix 3: A highly reliable, self-repairing operating system," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 3, pp. 80–89, 2006.

[10] "The Redox Operating System," https://doc.redox-os.org/book/, accessed: 2019-06-23.

[11] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from bell labs," *Computing systems*, vol. 8, no. 2, pp. 221–254, 1995.

[12] "DPDK Website," https://www.dpdk.org/, accessed: 2019-06-23.

[13] L. Gorrie *et al.*, "Snabb: Simple and fast packet networking."

[14] "Redox OS Drivers," https://gitlab.redox-os.org/redox-os/drivers, accessed: 2019-06-23.

[15] "Interrupt Handling in Ixy," https://github.com/tzwickl/ixy/tree/vfio-interrupt/, accessed: 2019-06-23.

[16] C. A. Thekkath, T. D. Nguyen, E. Moy, and E. D. Lazowska, "Implementing network protocols at user level," *IEEE/ACM Transactions on Networking*, vol. 1, no. 5, pp. 554–565, 1993.

[17] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 275–287.

doi: 10.2313/NET-2019-10-1_07

Peer-to-Peer Matrix

Quirin Heiler, Richard von Seck*, Jonas Jelten*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: q.heiler@tum.de, jelten@net.in.tum.de, seck@net.in.tum.de

***Abstract*—The current version of the Matrix network protocol relies heavily on the Domain Name Service and a Public Key Infrastructure, which stands in conflict with its overall decentralised design and makes it prone to censorship. This paper discusses a possible adaption to the Matrix APIs, which allows homeservers to operate in a self organized Peer-to-Peer manner, without the loss of any security assurances or the need for a centralized authority. The introduced approach will be based on the GNU Name System and tries to achieve a proper usability by allowing users to create human meaningful names for known conversation partners.**

***Index Terms*—matrix, p2p, gnu name system**

1. Introduction

The Matrix network tries to provide a generic and openly available service for message based communication. From the launch in 2014 [1], the network size has grown rapidly and is confirmed to have already reached a scale of more than one million users and 2500 homeservers by the year of 2017 [1]. The overall concept has thereby successfully proven to work in a real world, large scale application. However, at the current state of the project, the network is still inherently reliant on the use of the Domain Name Service (DNS) and consequently on a Public Key Infrastructure (PKI) to address and authenticate its homeservers. We believe that the removal of the dependence on these external, central authorities would benefit the system in several ways and comply with the projects core philosophies [2]. The key idea is that, given the ability launch Matrix homeservers in self a contained fashion and without having to rely on or even pay for an external service like DNS, more users will be able to operate their own servers, which would further increase the desired openness and decentralization of the network. Launching a personal homeserver comes with the additional advantage of allowing users to privately host their own conversations, thereby increases the control the user has over his personal data. Ultimately it is to mention that central authorities and DNS in particular [3] leave room for potential censorship, which should also be avoided when developing an open platform for communication.

This paper will propose an adaption to the current version of Matrix, which allows the operation of the underlying federation of homeservers in an open, in a fully decentralized and self controlled Peer-to-Peer (P2P) manner, without the need for a central authority. Additionally, it ensures, that our design fulfils the same standards of security and scalability as the original system, while also maintaining a reasonable degree of usability.

The introductory sections and the above mentioned statics are based on the well maintained documentation on the official matrix.org website [1], [2], [4]–[9]. The article about the Great DNS Wall of China, was published by Graham Lowe, Patrick Winters and Michael L. Marcus [3], it gives an example for the possible risk of DNS censorship. The central resource for this paper and the fundamental bases of the proposed concept was the master thesis of Martin Schanzenbach [10] about the GNU Alternative Name System. The habilitation of Christian Grothoff [11] about the GNUnet System contains the further descriptions of the GNU Name Service (GNS), which is the implementation of GADS, that we used in our design.

The paper will start by giving a basic introduction to the structure and functionalities of the Matrix network in Section 2., before Section 3. will discuss the challenges which have to be overcome when trying to find a suitable substitute for DNS. Section 4. and 5. give a short overview over GNS and present the actual design of our P2P network. The paper is concluded by a short introduction in available alternatives to GNS (Section 6.) and an evaluation (Section 7.) of the proposed solution.

2. Matrix

Matrix is an open standard developed by the Matrix.org Foundation. It defines a set of APIs with the goal to provide a free, globally available and decentralized service with no single point of control, allowing users to exchange persistent data in real time. The security of the system and the user's privacy is ensured through end-to-end encryption. Even though Matrix is still under development, there is already an implementation of the network available. Matrix tries to provide a simple generic interface, applicable to a variety of different use cases like instant messaging, IoT communication or to provide a signalling layer for webRTC based applications [4].

2.1. Basic functionality

The basic service provided by the Matrix network to an outside user is comparable to ordinary chat services. After generating an account at the Matrix homeserver

doi: 10.2313/NET-2019-10-1_08

of choice and acquiring a globally unique ID, users can start to access the system. All communication in Matrix is organized in rooms, which allow their members to publish/retrieve persistent data among/from all other users within. To enhance the user privacy, all room communication may optionally be end-to-end encrypted. Note however, that this communication is not limited to plain text messages. In fact Matrix can be used exchange arbitrary data between its users.

2.2. Network structure

While the current version of a matrix supports the use of webSockets and even includes a CoAP based ultra-low-bandwidth mode [4], we will focus on the default configuration and assume that all messages sent within the matrix network are HTTPS packages with JSON-objects (events) as their payload [4]. Going from there, the actual matrix network is formed between two main entities: Clients and homeservers.

2.2.1. Client Applications. Client [8] applications are important to the network in the sense that they represent the entry point for all user input. Aside from that, clients face a complete black box view of the actual Matrix network. Once connected to the user's homeserver, the further communication does not differ from the usage of contralised, server based web services. In that sense, the client only pushes events to the server as POST messages and stages pre-emptive GET requests to wait for answers. The crucial point is, that different clients in Matrix do never communicate to each other directly, even if they share the same homeserver.

2.2.2. Homeservers. The homogenous web of homeservers [9] is what actually makes up the network. They are responsible for storing their users' profile information and room state. Therefore every homeserver of all members of a room holds its own copy of the current room state. Since clients will only inform their own homeserver about new events, it is also the duty of every server to spread these events across all other servers in the room, while ensuring that every affected server will eventually obtain the exact same room state. This process of merging all incoming events of a room to a globally consistent state is called federation [9] and because it takes a vital role in the design of Matrix, the next section will be dedicated to explain the underlying processes.

2.3. Federation

This section will explain the algorithms behind the process of federation between multiple homeservers with the help of a simple example. We will assume a network layout as depicted in Figure 1.

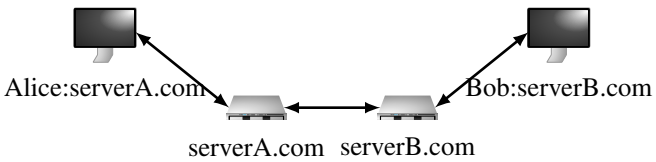


Figure 1: Network layout

The two users Alice:serverA.com (Alice) and Bob:serverB.com (Bob) try to exchange messages with the help of their respective homservers serverA (S_A) and serverB (S_B). The room they are using is assumed to be !roo:serverA.com (roo) (For more information about the naming scheme, see section 2.4). Each server models the room state as a directed, acyclic graph with a single root. We assume that the room was just created, hence the initial room state is empty (Fig. 2):

(a) Room state of S_A (b) Room state of S_B

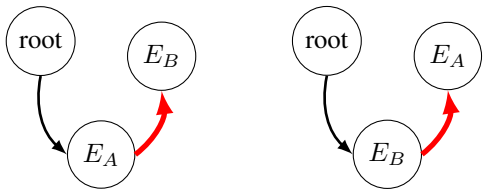
Figure 2: Initial state. Empty rooms.

Alice now pushes an event E_A to her homeserver. S_A examines its state graph and adds an edge from all existing events without an immediate child to the new event. In our scenario this results in exactly one edge from the root to E_A. Simultaneously, Bob pushes an event E_B to his homeserver. S_B reacts analogously to S_A. The resulting room state can be seen in figure 2:

(a) Room state of S_A (b) Room state of S_B

Figure 3: Room state after receiving first packages

Both homeservers now face the task of informing their respective counterpart about the newly received event. If they just forwarded the incoming messages to each other and treated the events received by other servers just like events from their own clients, both would end up with inconsistent versions of the state graph (Fig. 4):



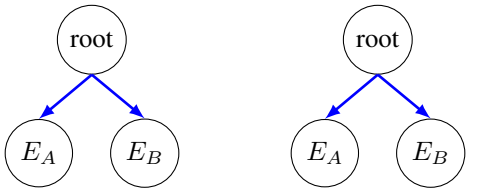
(a) Room state of S_A (b) Room state of S_B

Figure 4: Invalid federation. Inconsistencies marked red.

To circumvent this scenario, the servers do not only inform their counterparts about the existence of the new events, but also about the according parents of said event in the state graph. This will lead to the versions of the state graphs as depicted in figure 5.

Now each server holds a consistent copy of the room state and can pass the newly received events to their clients. However, this approach might not always be sufficient, as two messages in independent branches of the event graph might contain competing events. Imagine the events E_A and E_B in our example would both be attempts to rename the room to different names. To maintain a

doi: 10.2313/NET-2019-10-1_08



(a) Room state of S_A (b) Room state of S_B

Figure 5: Invalid federation. Consistent room states.

consistent room state, these events have to be handled identically at every involved homeserver. Specifically for situations like this, Matrix features so called room versions [5], which define a set of deterministic algorithms to resolve such situations. Different room versions can be applied to every room.

Note that the federation algorithm only ensures eventual consistency, but in no way the stability or immutability of any section of the graph, since new messages might have been delayed on their way through the network, they can reference any node in the state graph as their predecessor [5].

2.4. Discovery

As we already stated out, connections in the Matrix network are only established from client to server or from server to server. Therefore no network entity will ever have to resolve the IP address of a client before being able to build up a new connection. Servers on the other hand need to provide a static identifier as their name. This name can be either a static IP/Port-combination or a domain name with an according DNS Service record [9], [12]. User- and room-IDs are a triplet of three different values, containing: The address of a homeserver responsible for this user/room; A name which is locally unique within the scope of the homeserver; A specifier for the type of the address (i.e. room or user). This structure ensures that you will always be able to deduce the name, and with the help of DNS ultimately the IP-address of the server, responsible for a certain user or room, from the corresponding ID. To illustrate the concrete format, we will again look at the room from the example in section 2.3: !roo:serverA.com. In this case, "!" declares the type of the address and states out that it belongs to a room, while "roo" resembles the specific name of the entity and "serverA.com" is the address of the server where it was first allocated.

2.5. Third Party Integration

A key property of Matrix is that it does not simply provide a substitute for existing messaging platforms, but it creates possibilities to interoperate Matrix with other third party services.

2.5.1. Identity Service. The Matrix Identity Service API [7] allows users to create mappings from third-party identifiers like email addresses or phone numbers to Matrix IDs. These associations are managed by the so called identity servers. Other Matrix users can consult the identity servers to resolve a stored mapping, which allows to lookup a user's unknown Matrix ID from an already known third-party address. To ensure that users can only setup associations from third-party IDs that they actually own, the identity server provides a challenge which can only be solved by the owner of the given ID. In the scenario of creating a mapping from an email address to a matrix ID, the identity server would send an email containing a token to the given address. This token is then required to confirm the association.

2.5.2. Bridging. Matrix provides a whole variety of functionalities for the exchange of messages with third party services (e.g. email, facebook messenger, telegram, what's app, ...). So called Bridges [6] take the role of mediating between Matrix and third parties. Bridges can be designed as simple virtual users (bridge bots) [6], which take all received messages from one service and forward it to the other. More sophisticated approaches like Server-Server-Bridges [6] can be able to interconnect Matrix with other federated protocols (e.g. SMTP, SIP), by taking part in both server federations (matrix & third-party) and take the role of a general gateway between the two services. The overall advantage of Bridging is that it allows the usage of Matrix as a unified interface to communicate with arbitrary internet users, independent of what communication services they use.

3. Challenges

The central idea for our design of a P2P matrix is to keep the overall network structure and communication logics of matrix untouched and to only make minimal changes to the system. To achieve this, our approach will focus specifically on finding a reasonable P2P substitute for the DNS based naming system of Matrix. This section investigates the challenges and limitations, which arise when trying to build a fully decentralized naming system. Note that the considerations in the upcoming paragraph are based on [10] and are only applied to the context of Matrix.

Zooko's trilemma [13] is a hypothesis by Zooko Wilcox-O'Hearn, which suggests that there is always a trade off between three different special properties regarding the name system of a network protocol. To be more precise, these properties are decentralisation, memorability and security. Their meaning will be explained on the example of the current version of matrix:

Memorability. Matrix allows every user to freely pick a desired server name in the form of a web domain. As we expect that most users will prefer to pick simple and concise names, we can assume that these names are easily memorable.

Decentralisation. The name resolution process for server names is based on DNS, which is a fairly decentralised system. However, pure DNS does not impose any security features [10] like cryptographic authentication.

Security. To close this gap of security and to enable users to verify the authenticity of a DNS reply, additional mechanisms like TLS certificates [14] or the DNS Security Extension [15] are required, both of which rely on a Public Key Infrastructure (PKI) to issue, revoke and verify cryptographic certificates. The resulting hierarchy

of certificates, leads to a centralisation of trust [13], which originates from one or multiple trusted third parties. This means that in compliance with Zooko's trilemma, the gain in security is bought by sacrifices in decentralisation.

These considerations lead us to the following conclusion for the design of a P2P based Matrix system: If we want to no longer be reliant on a central, trusted third party, while also keeping Matrix resilient to adversaries, we will have to make sacrifices on the memorability of identifiers in order to increase the decentralisation of the network. However, keeping up a certain degree of memorability will be necessary for the usability of the system. The GNU Alternative Domain System (GADS) [10], is a fully decentralized naming system which was developed on exactly these considerations. The upcoming section will give a brief introduction into the GNU Naming System (GNS), which is a concrete implementation of GADS and part of the GNUnet alternative network stack.

4. GNU Name System

In GNS, every user is identified by a private-public key pair $(K_{priv}^{user}, K_{pub}^{user})$ and manages its own root zone. The ID (fingerprint) of a user's root zone is generated by hashing the public key of a user with SHA256 [16] (BASE32 [17] notation), it is considered globally unique. Zone files are comparable to their DNS counterpart and contain, among others, PKEY-records (reference the fingerprint of another zone file) and A/AAAA-records (for IPv4/IPv6). Each record of a zone file has a locally unique (A and AAAA records with the same name are allowed) name, picked by the owner of the zone. By creating additional key pairs, a single user can create and manage multiple zones. Users may sign their local zone files with the corresponding private key and make them publicly available for the network with the help of a censorship resistant Distributed Hash Table (DHT).

4.1. Name Resolution

The example network in figure 6 will be used to illustrate the address resolution process of GNS:

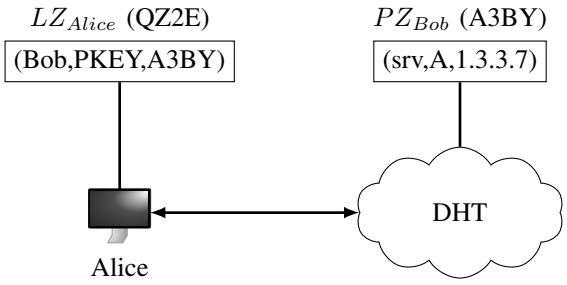


Figure 6: GNS example network state.

Bob wants to use his public zone file PZ_{Bob} with the fingerprint A3BY to advertise his server with the IP address "1.3.3.7". Therefore he creates an A record named "srv" in his zone file and uploads it into the DHT under its fingerprint "A3BY". Alice talked to Bob the other day and now she knows his zone's fingerprint. If Alice wants lookup the address of Bob's server, she has two equivalent options:

4.1.1. .zkey zone. With Bob's fingerprint in hand Alice can directly lookup Bob's server, analogously to DNS lookups by passing "srv.A3BY.zkey" to her GNS resolver. The ".zkey" Top Level Domain (TLD) is used to indicate that she wants to start the name resolution at the root of the global namespace. Thus the resolver will start by looking up Bob's zone file in the DHT, based on the identifier "A3BY" and afterwards retrieves the A record with name the name "srv".

4.1.2. .gnu zone. Unfortunately, the real fingerprint of a zone is way more complex than in our simple 4-digit example and therefore difficult to memorize. Luckily on the other hand, Alice has inserted a PKEY record named "Bob", which contains Bob's fingerprint in her local zone file LZ_{Alice}. She can now tell her resolver that it should start the address resolution process from LZ_{Alice}, by picking the ".gnu" TLD. The full GADS link would therefore be "srv.Bob.gnu". The resolver would now start by searching for entries in LZ_{Alice} named "Bob". Since "Bob" is a PKEY record the resolver will retrieve the contained finger print. The rest of the resolution process will then happen analogously to the ".zkey"-method.

In contrast to ".zkey"-addresses, which are valid globally, ".gnu"-addresses are only valid for a specific user, but at the same time allow the user to organize known own addresses by creating meaningful and easily memorable aliases.

5. P2P Matrix

This section presents our actual concept for establishing a fully decentralized, self organized, GNS based Matrix homeserver network (P2P Matrix). As a prerequisite we will assume that all clients and homeservers take part in the GNUnet system and thereby grant access to the GNS, while also providing bandwidth and storage for the GNS DHT.

5.1. Identification

We found that even in the current version of Matrix, it can be difficult to remember the address of a known user, because you do not just have to provide the actual username (which is possibly rather cryptic), you also need to know the name of the user's homeserver, which usually does not stand in any relation to the user. By introducing IDs that are created from public keys and therefore hard to remember, we expect this problem to get even more immediate. To address this issue, we want to minimize the contact a user has with these cryptographic identifiers and instead encourage the creation of personal aliases (pet names) for our system.

5.1.1. Homeserver IDs. Homeservers in P2P Matrix maintain a public GNS zone (Z_{server}). They are identified by their zone's fingerprint (F_{server}). This zone has to contain at least one A/AAAA record called "matrix", storing the IP address of the server. Every server can therefore be routed globally, by querying "matrix.FINGERPRINT_OF_THE_SERVER.zkey".

doi: 10.2313/NET-2019-10-1_08

5.1.2. Users IDs. When a user allocates a new account at a homeserver H. H will generate a public-private key pair ($K^{user}_{priv}, K^{user}_{pub}$) and publish a new zone file Z_{user} with the according fingerprint F_{user}. Z_{user} initially contains only one PKEY entry, which is called "home" and points to Z_H. Similarly to servers, users can now be globally identified by this fingerprint and their homeserver can be resolved by querying "matrix.home.FINGERPRINT_OF_THE_USER.zkey".

5.1.3. Room IDs. Similarly to user IDs, a room will also be associated with the fingerprint of a zone file containing a single PKEY entry pointing to the zone file of the server, which initially allocated the room (root). To distinguish room zones from user zones, this entry has to be called "root". As a result, the root server of a given room can now be looked up by querying "matrix.root.FINGERPRINT_OF_THE_ROOM.zkey".

5.2. Contact List

As already stated out in the preceding sections, every GNS user maintains at least one local zone file, which can be used to generate human readable mappings to known resources. Of course a user could now create mappings to known contacts in a private zone file at the client side, but since all the data in Matrix is stored at the homeservers and should be available even if the client logs in from another device, this approach would be rather insufficient.

"Remote local zone file". To address this issue, we will expand the Matrix client-server interface, by operations which allow a user to read and manipulate the Z_{user} file, stored at its homeserver. The user can therefore create easy to remember aliases which will then be resolved locally by the homeserver. The following example illustrates this concept (figure 7):

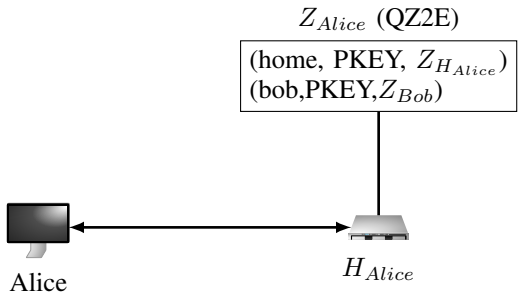


Figure 7: Example remote local zone file

Alice already knows Bob's ID and set a PKEY record named "bob" in her "remote local zone file" Z_{Alice}, located at her homeserver H_{Alice}. From this point on Alice will no longer need to specify Bob's ID when talking to her homeserver. Because the server knows that it is talking to Alice, which is identified by her zone file fingerprint Z_{Alice}, it can resolve the address of Bob's homeserver by looking up "matrix.home.bob.Z_{Alice}.zkey". The GADS resolver will realise, that the zone file Z_{Alice} is already present locally and will not have to resolve this first zone over the network. Bob's ID on the other hand can be read directly from the according PKEY entry in Z_{Alice}.

Note again, that GNS also allows users to create private records in their local zone files, which will not be published. As Alice probably does not want to share all her contacts with the entire network, she would be able to only set private records in Z_{Alice} (excluding the home record). Nevertheless, since the homeserver only performs local lookups on her zone file, this change would not impair the overall functionality.

By the introduction of the "remote local zone file", we enable users to maintain a listing of known IDs and rename them to locally unique and easily memorable aliases, which can be accessed even when accessing the homeserver from different devices. While the above example only mentioned user IDs, the concept works for room IDs analogously.

6. Alternatives to GNS

The concept of introducing a fully decentralized alternative for DNS is not entirely new to the domain of P2P networking. To serve as a bases for upcoming comparisons, this section will introduce two well known P2P alternatives to our GNS based approach.

6.1. TOR

".onion"-addresses are used by the TOR [18] network to advertise hidden services. Conceptually, they are comparable to GADS ".zkey"-addresses, they follow the form "CRYPTOGRAPHIC_IDENTIFIER.onion" and are advertised over a Distributed Hash Table. Note, that each such DHT entry does not contain a full zone file, but only an individual IP address. ".onion"-addresses do not provide a way for users to create human memorable aliases. In that sense, they might be considered a minimal archetype of P2P naming systems.

6.2. Invisible Internet Project

The Invisible Internet Project [19] (I2P) differentiates between two address types. The basic type works similar to the aforementioned ".onion"-addresses, with the only difference that the top level domain is ".i2p" instead. The second type are locally valid aliases, which can be set by the user and are being stored in the so called address book. Aside from being able to manually read and write entries to their own address book, users can also subscribe to public address book files of other users. The latter feature allows users to discover new addresses without ever coming in contact with the underlying cryptic identifiers.

7. Evaluation

This section evaluates the P2P matrix design, established in Section 5, against the initially proclaimed properties of security, usability and scalability.

7.1. Security

The presence of malicious notes in open P2P networks is an unpleasant, yet unavoidable truth. Consequently, it is vital for our system to prove resilience against a variety of different attacks.

doi: 10.2313/NET-2019-10-1_08

7.1.1. Authenticity.

Secure Addresses. Since Matrix can be used to exchange highly confidential data, the naming system has to allow users and servers to safely authenticate their communication partners. This is ensured in the original Matrix with the help of cryptographic certificates, which are issued by an external certification authority. All of the introduced P2P naming systems on the other hand do not require such certificates, since the information necessary for validating cryptographic signatures of a domain owner is already contained in the cryptographically generated domain itself. In that sense, GNS/TOR/I2P addresses can be considered secure by design [10]. To put this into the perspective of the previous considerations on Zooko's trilemma: The described P2P services are sacrificing the memorability of their domains to achieve a high level of security and decentralization. TOR hidden services demand similar authenticity assurances as ordinary websites. The large scale deployment of TOR and its hidden services can therefore be considered a proof of this concept of self certifying addresses.

Fuzzy fingerprinting. The preceding considerations about authenticity in ".onion"-addresses are based on the assumption that every address is bound to a globally unique key pair and can therefore not be claimed by any entity other than the actual owner of said key pair. However, this assumption does no longer hold for partially matching addresses. As a result, it is a practice for attackers to generate ".onion"-addresses with a similar appearance to an actual address [20]. The aim of this attack is to abuse the user's laziness, when comparing the attacker's address to the original and thereby trick the user into inadvertently accessing the attackers address. Notice that this attack becomes increasingly effective if the user frequently accesses the same service and assumes to recognise its correct address by just quickly skimming it.

While an attacker could still try to perform a similar attack in our P2P Matrix, it is expected to be way less effective. A custom pet name system like in GNS or I2P, helps to minimize the risk for users to fall for this attack, since it enables the user to access the given address without the need of constantly supplying its cryptic and easily mistakable identifier, thus avoiding the risk to mistake it with a similarly looking ID [10].

7.1.2. Censorship. The strong authenticity assurances of GNS, combined with the randomized routing algorithm and the redundant storage of entries of its R5N [21], [22] based DHT make it difficult for attackers to manipulate or deny access to other user's zone files [11], [21]. This property makes P2P Matrix resilient to potential censorship by manipulation or denial of its naming system.

7.2. Usability

An important goal of our design was to maintain a high level of usability by minimizing the exposure a user has with the cryptographic GNS IDs, yet providing an understandable naming scheme. As a result, every Matrix entity can be addressed with only a single GNS ID, while the corresponding addresses of an entities homeserver (see sections 5.1.2 & 5.1.3), provide the same information as addresses in the original Matrix, as they: Provide a globally unique identifier for the entity; Resolve to the IP address of the responsible homeserver; Hold type information (i.e. "home" for users/ "root" for rooms). This structure even yields advantages to the current version of Matrix, where users have to memorize pairs of arbitrarily chosen server and user/room IDs. The introduced pet name system allows users to setup easily memorable aliases for already known conversation partners and further enhances the usability.

However, this simplification will not improve the overall discovery of previously unknown users/rooms. Especially the field of verbally exchanging ID information is expected to suffer from the newly introduced, hard to memorize GNS identifiers. The impact of this problem could be reduced by encouraging to exchange public key information with the help of mobile devices and technologies like QR-codes or by using the Matrix Identity Service to create mappings from third party addresses (e.g. email) to Matrix IDs. While the latter option would allow users to reach a user experience comparable to the current version of Matrix, it can not be considered an optimal solution, since P2P Matrix is aimed to function without the dependence on third party services.

7.3. Scalability

An important aspect, which is yet to be covered is the question whether GNS yields the ability to work in real world, large scale environments. R5N is an extension of the Kademlia [23] algorithms and inherits the capabilities of dealing with the dynamics of leaving and joining peers in large scale networks [24] (churn). The R5N routing algorithm allows looking up and depositing zone files in the time complexity $\mathcal{O}(\sqrt{n} * \log n)$ [21](where n is the number of peers in the network), thus manages to maintain efficient lookups, even in large scale networks.

8. Conclusion and Future Work

The central goal of the Matrix network to build a decentralized platform for communication which gives the user maximum control over his personal information align with the benefits which would arise from rebuilding matrix into true P2P system, which does not rely on central authorities. This paper introduced the design for such a P2P matrix, which utilizes the GNU Naming System as a secure and scalable alternative to the Domain Name System to create a simple, hierarchical naming scheme. Even though the approach focused heavily on the use of a pet name system to maintain a high usability, the effects of using "randomly generated" IDs will be noticeable by the users (especially without making use of 3rd party IDs and the Matrix Identity Service), as it will become more difficult to discover new users within the network.

A proposal for a future extension of the introduced design could be to reduce this problem by allowing users to find new contacts within Matrix itself. Following a simple "friends-of-friends" logic it can be assumed that users are likely to interact with the contacts of their

own contacts. Users could be allowed to subscribe to the contact list of other users in the style of I2P address books, in order to automatically accumulate new contacts without relying on cryptographic identifiers at all.

To actually find out how our design works in a real world application and whether it resembles an improvement to the current state of Matrix, it would be the next logical step to actually build and test a minimal prototype. Since both services follow a similar internal structure, it should even be rather simple to create a Server-Server-Bridge between the official version of Matrix and a P2P prototype to run them in full interoperability.

References

[1] "Matrix statistics," https://web.archive.org/web/20190515055721/https://matrix.org/blog/2017/07/07/a-call-to-arms-supporting-matrix/, accessed: 2019-05-15.

[2] "Matrix manifesto," https://web.archive.org/web/20190808012751/http://matrix.org/foundation/, accessed: 2019-08-08.

[3] G. Lowe, P. Winters, and M. L. Marcus, "The great dns wall of china," *MS, New York University*, vol. 21, p. 1, 2007.

[4] "Matrix Main Page," https://web.archive.org/web20190620111226/http://matrix.org/, accessed: 2019-06-20.

[5] "Matrix specification," https://web.archive.org/web/20190611230822/https://matrix.org/docs/spec/, accessed: 2019-06-11.

[6] "Matrix Types of Bridges," https://web.archive.org/web/20190803182830/https://matrix.org/docs/guides/types-of-bridging/, accessed: 2019-06-20.

[7] "Matrix Identity Service," https://matrix.org/docs/spec/identity_service/r0.2.1, accessed: 2019-08-08.

[8] "Matrix client-server-api," https://web.archive.org/web/20190808012751/https://matrix.org/docs/spec/client_server/r0.5.0, accessed: 2019-08-08.

[9] "Matrix server-server-api," https://web.archive.org/web/20190808012750/https://matrix.org/docs/spec/server_server/r0.1.3, accessed: 2019-08-08.

[10] M. Schanzenbach, "Design and implementation of acensorship resistant and fully decentralizedname system," 2012.

[11] C. Grothoff, "The gnunet system," Ph.D. dissertation, 2017.

[12] A. Gulbrandsen and L. Esibov, "A dns rr for specifying the location of services (dns srv)," 2000.

[13] "Zooko's trilemma," https://web.archive.org/web/20040616080110/http://zooko.com/distnames.html, accessed: 2019-08-28.

[14] P. Saint-Andre and J. Hodges, "Representation and verification of domain-based application service identity within internet public key infrastructure using x. 509 (pkix) certificates in the context of transport layer security (tls)." *RFC*, vol. 6125, pp. 1–57, 2011.

[15] M. Larson, D. Massey, S. Rose, R. Arends, and R. Austein, "Dns security introduction and requirements," 2005.

[16] T. Hansen, "Rfc 6234-us secure hash algorithms (sha and sha-based hmac and hkdf)," 2011.

[17] S. Josefsson, "The base16, base32, and base64 data encodings," 2006.

[18] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the tor network," in *International symposium on privacy enhancing technologies symposium*. Springer, 2008, pp. 63–76.

[19] F. Astolfi, J. Kroese, and J. Van Oorschot, "I2p-the invisible internet project," *Web Technology Report*, 2015.

[20] "Fuzzy Fingerprints - Attacking Vulnerabilities in the Human Brain," https://github.com/vanhauser-thc/THC-Archive/blob/master/Papers/ffp.pdf.

[21] N. S. Evans and C. Grothoff, "R5n: Randomized recursive routing for restricted-route networks," in *2011 5th International Conference on Network and System Security*. IEEE, 2011, pp. 316–321.

[22] C. Grothoff, "The gnunet dht."

[23] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[24] Z. Ou, E. Harjula, O. Kassinen, and M. Ylianttila, "Performance evaluation of a kademlia-based communication-oriented p2p system under churn," *Computer Networks*, vol. 54, no. 5, pp. 689–705, 2010.

doi: 10.2313/NET-2019-10-1_08

A Comparison of OPC UA vs. VSL for IoT

Tobias Leibbrand, Christian Lübben*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: tobias.leibbrand@tum.de, luebben@net.in.tum.de

***Abstract*—The basis of the Internet of Things (IoT) is Machine-to-Machine (M2M) communication. In order to enable this communication, the machines must know on what basis they can communicate with each other. In order to standardise this communication across platforms middleware is used. In the following two different approaches are compared, the OPC Unified Architecture (OPC UA) and Virtual State Layer (VSL). OPC UA follows the client-server approach, whereas VSL builds on peer-to-peer technology.**

The two standards are compared regarding accessing, storing, discovering and transporting data as well as due to data structure and security aspects. The comparison shows that OPC UA is particularly suitable for observation tasks and VSL when communication between all devices in the network is required.

***Index Terms*—OPC UA, VSL, IoT Middleware**

1. Need of IoT Middleware

According to research from the Statista Research Department, more than 30 billion devices around the world will be connected to the internet in 2020 [1]. To give that some perspective, that is more than three devices for every person on earth. The Internet of Things (IoT) is indispensable. With this number of devices it goes without saying that not all of these devices can be controlled by humans, but the devices also have to communicate with each other by using machine-to-machine (M2M) communication. In order to enable the communication of devices from different manufacturers, it is necessary to create special interfaces. This task is fulfilled by IoT middleware frameworks. In the following, two different approaches, OPC Unified Architecture (OPC UA) and Virtual State Layer (VSL) are compared to each other, to discuss the pros and cons and to elaborate on the application purposes.

OPC is an industry standard that has been developed in close cooperation with the automation industry and is now implemented by almost all major companies [2]. It enables secure and lossless data exchange and is platform independent since the introduction of the Unified Architecture extension. The main advantage of standardization is that devices from different manufacturers can communicate with each other out of the box [3]. But it is more than just a protocol for data exchange. OPC UA also specifies the rules for communication between computers or devices. In this client-server-model, there are two roles that a communication partner can assume. These two partners communicate with each other, whereby the establishment of the connection can only be started from the client. He can make inquiries to the server and then receives an answer or he can make a subscription to be notified about changes. This construction is called service-oriented. There is no restriction on how many clients can talk to one server or how many servers can be connected to one client. No matter what the configuration is, clients send message requests to servers and servers respond. OPC UA also does not specify how two servers can communicate with each other, therefore horizontal communication is not natively supported.

Virtual State Layer (VSL) has been designed and developed by the team led by Marc-Oliver Pahl at the Technical University of Munich (TUM). In contrast to the OPC approach it does not pursue the classic client-server approach but follows the approach of peer-to-peer networks which has become increasingly popular in recent years. It was designed specifically in order to enable distributed Smart Space Orchestration (S2O). Therefore, the framework became very much data-centric. Fully focussing on this concept VSL offers a full separation of service logic and data. [4]
The network is made up of Knowledge Agents (KA). These take control over all connectivity tasks and data access in the background. The data access is designed to be fully transparent. Therefore the data access can take place in such a way, as if these would be locally available on the current host. VSL also fully decouples data producers and data consumers. This allows completely new approaches in development. As a consequence a hardware sensor is independent of the orchestrating control software and does not have to run at the same time. This increases the robustness and possibly the energy efficiency of IoT nodes. [4]

The aim of this work is a comparison of the M2M-communication standards OPC UA and VSL. Therefore, important aspects of the communication standards are used to obtain a basis for comparison. The selected aspects are: data access, data structure, data storage, data discovery, data transport, and security.

This work is structured as follows. Section 2 mentions other major work dealing with one of the two standards. Section 3 deals with how existing data can be accessed. Section 4 focuses on the data structure. Datastorage is discussed in section 5. Chapter 6 describes how the recognition of new devices and their provided data works. Chapter

doi: 10.2313/NET-2019-10-1_09

7 introduces the protocols used and Chapter 8 introduces the security mechanisms. The conclusion shows which specification is more suitable for which use case.

2. Related Work

Since OPC UA is a widespread communication standard, there are all kinds of documentation and textbooks such as [5], [6] and [7]. These describe the OPC UA standard in all details and also give practical application tips. There are also numerous papers dealing with this standard. Article [8] gives a short overview which aspects of OPC UA are relevant. Paper [9] analyses the standard in terms of performance and paper [10] compares approaches with other IoT middelware approaches. Paper [11] discusses possible extensions of the OPC standard to enable bidirectional communication.

Since VSL is an in-house development of the Chair of Network Architectures and Services and has not been on the market for long, there are only publications on this topic from the Chair itself, such as the doctoral thesis [12] of Dr. Marc-Oliver Pahl and some papers like [4] and [13] to which he contributed.

3. Data Access

Since data exchange is a central component of IoT communication, this chapter is considered right at the beginning. In the following we will consider the possibilities available for interacting with data in the network.

3.1. Data Access in OPC UA

Within the Client-Server-Architecture the classic way to query or write data is by using the Read and Write OPC UA services, which enable an OPC UA Client to read and/or write several attributes of nodes and are focused on bulk read/write operations. To be notified when a value is changed subscription methods are available. Within a subscription data are transported to the client contingent on events or evaluations of data and data changes. For this purpose an OPC UA Client has the ability to create monitored items in the OPC UA Server. Those items monitor AddressSpace Nodes and their real world counterparts. Fig. 1 shows the relationship of monitored items and nodes in the AddressSpace on the one hand side and to a specific subscription on the other hand. [9, p. 166f.]

3.2. Data Access in VSL

In VSL the data access is handled transparently and offers a data-centric, semantic interface as Application Programming Interface (API). The addresses in VSL are structured in a hierarchical tree structure. An address that supports worldwide distribution looks like this vsl://[siteID]/[kaID]/[serviceID]/[subNodeAddress]/. The distributed VSL IoT spaces are connected and are identified by the siteID. The servers that take over the task of data distribution are the Knowledge-Agents and are identified over the kaID. Several services can be connected to one KA which are independent from each other and are identified by the serviceID. All identifiers are unique.

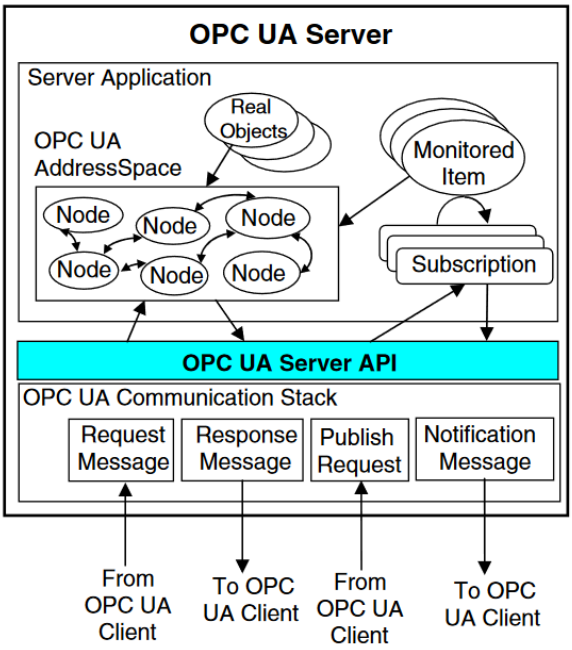


Figure 1: OPC UA Server [9, p. 166]

The rest of the address is also inheritance hierarchical and depends on the respective service. The data can be accessed via get and set calls and it is possible to subscribe to changes. [4]

3.3. Comparison Regarding Data Access

Data access is structurally the same for both frameworks. Data can be queried, data can be changed or subscreened for changes. However, the effectiveness of the individual queries differs considerably. OPC UA is primarily designed to query complex data structures, which means that if only one value is to be queried, all other values still have to be passed. VSL, on the other hand, is designed to process individual sensor data and therefore has no problems and is therefore more efficient.

4. Data Structure

In addition to data access it is of course also important how the data structure is organized in order to create the balancing act between clarity and flexibility.

4.1. Data Structure in OPC UA

The special thing about the OPC UA is, that it is reasonably flexible as well as simply structured compared to other approaches. The basic element is a node which is simply a highly structured data consisting of a set of predefined attributes and relationships. Using this very simple data element, an OPC UA address space can be created allowing for very complicated processes to be represented. The flexibility of the address space allows a designer to present not only raw process data, but also extensive information about the state of the underlying process and the process environment. This flexibility ensures that even

doi: 10.2313/NET-2019-10-1_09

complicated systems can be exposed using OPC UA. It also enables that every device could use a different data structure. [7]

4.2. Data Structure in VSL

The overall complexity is reduced significantly by having one data type per semantic functionality. An object-oriented information model is used to enable complex data objects and still keep them flexible enough. These models are called VSL Context Models. A library with basic data types is already available. The developer can create a suitable model for each service, which can then be reused for similar services in the future. The Context Models are stored in a global Context Models Repository, which distributes them once when the network ist initialised. Once the network is running, each KA stores them in a local context models repository which is kept synchronous with the other KA. [4]
All data can only contain the actual state and no time series data. Therefore, for example, it was not sufficient to have only one date for a lamp, whether the light is on or off, because the control logic of the lamp never knows when the status was changed. Also, it is a problem that the digital status can deviate from the real status. Therefore it is necessary to add a desired status to the lamp so that the logic can compare and set the current status to match the real event.

4.3. Comparison Regarding Data Structure

Both approaches are very similar here as both are based on an object-oriented information model. This approach allows a good standardisation of as many variants as possible without losing the necessary flexibility. The major difference is that OPC UA comes with a lot of models, whereas VSL only provides the developer with the basic data types and gives him a free hand to specify them. Over time, a publicly accessible and comprehensive Context Model Repository could compensate for this.

5. Data Storage

If large amounts of data are to be processed or data histories are to be created, it is interesting to know whether the data is stored locally or has to be requested each time via the network.

5.1. Data storage in OPC UA

An OPC UA server stores all data locally. These data can only be queried by connected clients and cannot be viewed by other servers. In addition to the current live data, a historical dataset can be implemented, which logs the live data. A simultaneous working on the same dataset is not provided.

5.2. Data Storage in VSL

In VSL, data is always stored locally with the respective Knowledge Agent. To have the data prompt or as close as possible is advantageous for performance reasons. Because of the data transparency it can be queried at every other Knowledge Agent like locally available. For reasons of consistency, data cashing has not been used in the current implementation.

5.3. Comparison Regarding Data Storage

Although the way both implementations regulate data access is slightly different and VSL provides distributed access to the data, none of the implementations provides an inherent ability for long-term archiving of the data. This task must be done by an extra client that is subscribed to changes.

6. Data Discovery

Since networks are adapted or renewed according to their requirements with varying frequency, it is interesting to see how new devices and their data are handled in the network. If the active devices are changed with a high frequency, it would therefore be cumbersome if they had to be configured manually each time, whereas it would not be a problem with only a few devices.

6.1. Data Discovery in OPC UA

In OPC UA every service offered by a server will be represented as an endpoint. An endpoint is a connection to a device that offers some specific functionality that is sometimes only available through that specific connection.

Any server that wants to offer a service opens a discovery port for messages from the client after booting. When a client wants to connect to a server, it scans for the servers discovery endpoints and filters out the appropriate server. The service discovery endpoints data also contain transport and security information [7]

6.2. Data Discovery in VSL

In the IoT area it is not untypical to work with constantly changing devices. Therefore the devices in VSL are not addressed directly, but typed via the offered service. A node can consist of several types. The KA can search for these types at runtime in order to determine whether new devices have been added to the network.The typesearch is handled by the VSL and automatically is forwarded to the relevant KA. The information about all available data nodes is periodically exchanged in the background, which accelerates the search because it can be done locally. [4]

6.3. Comparison of Data Discovery

On this point the two standards differ due to the different applications. In the OPC UA, the service detection serves mainly the one-time configuration during the installation of the network. Therefore it doesn't matter if discovery process is not so efficient. In VSL, on the other hand, service detection has a much more central significance, since the network is increasingly designed for interchangeable components.

doi: 10.2313/NET-2019-10-1_09

7. Data Transport

Besides the structural points, it can also be interesting which protocols were used to implement the corresponding standard. This is particularly important if, in addition to the local implementation, external access via the Internet is also required.

7.1. Data Transport in OPC UA

There are currently two protocols, as well as a mixed variant that combines both of them. All variants can be used in parallel without any functional disadvantage. The standard protocol UA-Binary, which all OPC UA implementations must support, is the binary protocol. This offers the best performance, because it has a low overhead and is specified exactly, and consequently has only few degrees of freedom. The second variant XML-SOAP has the advantage that it is firewall friendly, because the communication works over HTTPS. It is also easier to process the received data for implantation. Since this variant has a higher overhead, however, it has almost no acceptance with embedded devices. The hybrid variant combines the advantages of both protocols by binary coding of the payload in the HTTPS frame. The ANSI-C stack implemented by the OPC Foundation supports the UA binary protocol and the hybrid protocol. The standard protocol should be the efficient binary protocol and only in special cases the hybrid protocol should be used. The Web service implementation is available for applications that require Web services. [14]

7.2. Data Transport in VSL

The Knowledge Agents in VSL have the task of maintaining the network structure and exchanging data. To be compatible with other approaches, connectivity is implemented as peer to peer overlay. VSL uses multicast to maintain the structure, and unicast for direct data exchange. The Transport Manager, Connection Manager, and the Overlay Manager maintain the P2P overlay. The entire inter-node connectivity is encapsulated in these modules. The transport manager in the current implementation uses standard protocols such as HTTP over TCP/IP as transport, but this can easily be exchanged with other communication protocols. However, all protocols used are interchangeable, so that scaling is not an obstacle. [4]

7.3. Comparison of Data Transport

Both types of implementation do not really differ, as they use standard protocols for large parts of the communication. Since the protocols have been implemented interchangeably, it is always possible to weigh universal applicability against performance.

8. Security

Since sensitive data is also always exchanged in the Smart Home sector, it is very important that appropriate security mechanisms are implemented. Which ones are available will be discussed in the following.

8.1. Security in OPC UA

Safety in OPC UA ist based on multiple layers. UA Security is a multi-layered concept. The most important protection goals such as authentication, authorization, encryption and integrity are maintained. Access at the application level is ensured either by using certificates or by logging in with a user name and password. Access rights can be assigned group-specifically. In order to perform intrusion detection, all login procedures can be logged. At the transport level, corresponding algorithms from the Web communication are used for encryption. However, the most secure specifications are of no use if they are not implemented or only partially implemented by the manufacturer. To avoid this, every OPC UA certified product must meet these specifications. [15]

8.2. Security in VSL

The special thing about VSL is that it is built on a self-organizing P2P network, unlike other data-centric IoT designs. Access control for read and write actions can be specified for each class of data nodes. To establish a secure communication between the KA, X.509v3 certificates are used in the current implementation, which establish a TSL connection. This connection is used to exchange the keys required for data encryption. Each service and each KA has its own certificate which is automatically exchanged in the background. [4]

8.3. Comparison Regarding Security

Security is a top priority in both approaches. Both offer secure data transfer as well as access control through authorization. The prevention of unauthorized data access is also guaranteed. As a weak point one could consider that access control can be deactivated with OPC UA, which may become a security problem with careless handling.

9. Conclusion

Both frameworks were designed to enable machine-to-machine communication. There are no major differences with regard to data structure, data storage, data transport, and security. However, the fact that both approaches were designed for different purposes can be seen not only in the different details when comparing the approaches, but also in the chosen type of network communication.

The implementation of OPC UA with its client-server-model, on the one hand, is ideally suited for monitoring processes. The well-structured object model enables a smooth interaction of industrial plants from different manufacturers without having to invest a lot of time and money in the installation. With its holistic data objects, the protocol is ideally suited for monitoring and controlling process sequences. Data access is primarily designed to query complex data structures and is less efficient in processing individual (sensor) data. Data and device discovery is not the major task as OPC UA is used in the context of one-time configurations. Therefore an inefficient discovery process is not a major issue. There is still room for improvement in direct communication from server to

doi: 10.2313/NET-2019-10-1_09

server, since there is no standardised communication for this, but only an improvised solution with a bundle of clients and servers on both sides.

Peer-to-peer communication at VSL, on the other hand, enables horizontal communication between the individual Knowledge Agents. This makes it possible for the servers to communicate with each other, thus enabling Distributed Smart Space Orchestration. This means that the actual sensors can be separated from the logical programming and can therefore be operated in an energy and cost-efficient way, optimal for Smart home implementations. Data access is designed to process individual (sensor) data efficient and the service detection has a much more central significance as the network has to work with a growing number of interchangeable components.

In short, the two implementations OPC UA and VSL cover their respective intended tasks very well. Further research could explore those theory based hypotheses empirically.

References

[1] S. R. Department, "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)," Tech. Rep., 2016, https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[2] OPC Foundation. Major Automation Industry Players join OPC UA including TSN initiative. 2019-08-31. [Online]. Available: https://opcfoundation.org/news/press-releases/major-automation-industry-players-join-opc-ua-including-tsn-initiative/

[3] OPC Foundation. What is OPC? 2019-06-19. [Online]. Available: https://opcfoundation.org/about/what-is-opc/

[4] M.-O. Pahl and S. Liebald, "Information-Centric IoT Middleware Overlay: VSL," in *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Garching b. München, Germany, Mar. 2019.

[5] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.

[6] J. Lange, F. Iwanitz, and T. J.Burke, *OPC - Von Data Access bis Unified Architecture*, 5th ed. Berlin, Offenbach: Vde Verlag GmbH, 2013.

[7] J. S. Rinaldi, *OPC UA Unified Architecture - The Everyman's Guide to the Most Important Information Technology in Industrial Automation*, 1st ed. CreateSpace Independent Publishing Platform, 2016.

[8] S.-H. Leitner and W. Mahnke, "OPC UA - Service-oriented Architecture for Industrial Applications," *Softwaretechnik-Trends*, vol. 26, 2006.

[9] F. C. Salvatore Cavalieri, "Analysis of OPC UA performances," *Computer Standards & Interfaces*, 2013.

[10] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, Feb 2019.

[11] D. Torben, P. Florian, G. Sten, and P. Julius, "Bidirektionale Kommunikation mit OPC Unified Architecture," *Softwaretechnik-Trends*, vol. 26, 2016.

[12] M.-O. Pahl, "Distributed Smart Space Orchestration," Ph.D. dissertation, Technische Universität München, München, 2014.

[13] M.-O. Pahl, S. Liebald, and C. Lübben, "DEMO: VSL: A Data-Centric Internet of Things Overlay," in *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Garching b. München, Germany, Mar. 2019.

[14] ascolab GmbH. OPC UA Protokolle. 2019-06-19. [Online]. Available: http://www.ascolab.com/de/unified-architecture/protokolle.html

[15] ascolab GmbH. OPC UA Sicherheitskonzept. 2019-06-19. [Online]. Available: http://www.ascolab.com/de/unified-architecture/sicherheit.html

Quality Enhancement in Written Examinations by Automatic Recognition of Correction Results

Arian Mehmanesh, Stephan Günther*, Johannes Naab*, Maurice Leclaire*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: arian.mehmanesh@tum.de, guenther@tum.de, naab@net.in.tum.de, leclaire@in.tum.de

***Abstract*—An examination score determined by human correctors can be erroneous in multiple ways. In this case the focus is on errors caused by miscalculating the score manually. The goal is to estimate a rate of exams with calculation mistakes and determine which factors are likely to increase this error rate when designing an exam. To achieve this, 525 sample exams are digitized by hand with help of an automated system detailed herein. Afterwards, the digital exam scores are scanned for errors automatically. Statistical analysis of these errors yields the following results:**

The quote of exam points miscalculated in this sample is determined to be 4.2%, resulting in a confidence interval between between 2.7% and 6.4% (95% conf. level). On average, the correctors made an error when a problem has 10.4 subproblems distributed over 3.9 pages. The mean impact of an error is 0.8% of the total credits. Considering the time it takes to manually add the points and determine the grade, in combination with this error rate, automated score calculation systems for exams are proposed as a solution.

***Index Terms*—examinations, human error, correction, miscalculation, scoring**

1. Introduction

Human error is unavoidable in the correction of examinations by hand. While these faults can occur by overlooking correct answers or grading similar responses differently, the last step of every evaluation is to sum all points and determine the final grade. This paper focuses on quantifying the errors made in calculating the exam scores since they are measurable and comparable across all forms and topics of examinations. These arithmetical errors are also chosen because they are computationally detectable.

To achieve this, a sample examination is analyzed in its entirety for falsely calculated credits. This analysis requires an optimized interface to facilitate quick acquisition of the recognized scores. After the recognition of handwritten exam scores the correctors' faults are detected by an automated validation algorithm.

After all errors are found, the percentage of miscalculated exams and by how much the error deviates from the correct credits on average are of interest. Additionally, relations between the error rate of a problem and the count of its subproblems (or how many pages the problem spans, respectively) are inspected.

The content of this paper is structured as follows: In Section 2 a related study is presented. The dataset used in this project is detailed in Section 3.1, followed by a description of the software used to extract numerical score values from exam images in Section 3.2. These values are statistically analyzed in Section 3.3. The results of this analysis are presented in Section 4, amounting to the conclusion in Section 5.

2. Related work

Studies discussing errors in examinations are common, but for comparison to this analysis they are required to differentiate between score calculation mistakes and other errors. Phillips & Weathers have analyzed 5017 standardized tests (Stanford Achievement Test) in 1958 [1]. They quantified and distinguished different types of errors, like correctors not following the instructions or falsely computing the final grade based on the students total score. The focus of this paper, the incorrect summing up of scores, was observed aswell but referred to as "counting error". Out of the total 5017 tests, 630 of them were miscounted (13 %). This was the most prominent fault, causing 45 % of all errors.

3. Methods

After the description of the dataset used for this project, the two main parts of the methodology are detailed. They consist of the interface used for recognizing the written scores and how these determined values are analyzed for errors.

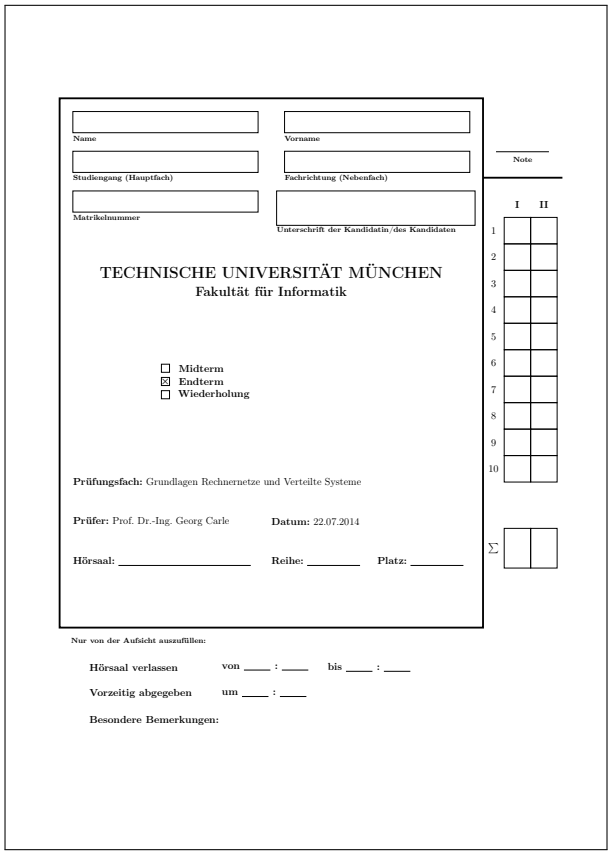
3.1. Dataset

The analyses herein are based on a digitized endterm examination of 2014 provided as scanned images. It was held at the Technical University of Munich (TUM) on the topic of "Basics in Networking and Distributed Systems" [2], consisting of 525 individual exams.

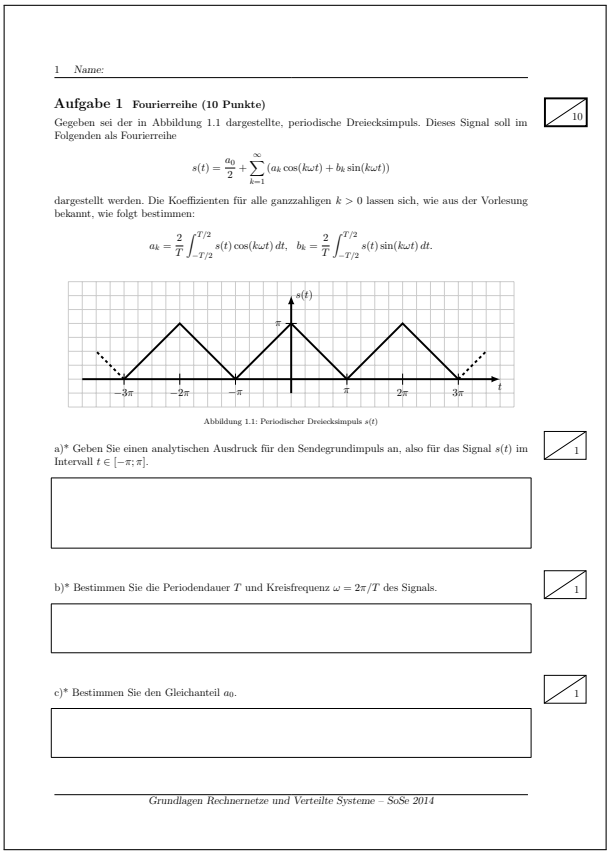
Figure 1a shows the front page of the exam, the points noted here sum up to yield the final grade. The first problem of the exam is demonstrated in Figure 1b, where the scores of the subproblems are added and are written in the top box. This result of problem 1 is carried over to the front page (Fig. 1a).

Every exam consists of 68 score boxes, resulting in 35700 total boxes available. This examination was evaluated twice by the correctors, in a first and second run. The result of the second run determines the final grade.

doi: 10.2313/NET-2019-10-1_10



(a) Front page score boxes and total sum.



(b) Problem score box with subproblems.

Figure 1: Sample pages from the studied exam.

3.2. Recognition

The optical character recognition (OCR) of the written credits is performed manually. Automated OCR or interpretation by a machine learning approach exceeds the scope of this analysis. To optimize this process it is necessary to automate the displaying of score boxes to the reviewer and recording the recognized score for each problem. Additionally, metadata should be tracked for every box, such as the page on which the box was located and the time interval it took the reviewer to recognize and enter its numerical values. For the implementation of this automation the programming language Python [3] is used due to its ease of use and legibility.

The structure of the program can be reduced to the Model-View-Controller pattern [4] which allows these three components to be detailed separately.

3.2.1. Model. The model replicates the dataset and consists of the whole `ExamBatch`, a single `Exam` and the individual `Problem` which represents a score box. An `ExamBatch` manages a list of `Exams`, whereas an `Exam` stores a tree of `Problems`.

In the example of Figure 2, the root node of the tree is the score box of the total exam credits. Subordinated are `Problem` 1 and 2 on the front page, which are the scores of `Problem` 1 and 2 carried over from the inside of the paper exam sheet. Each `Problem` contains two subproblems.

The tree structure is chosen because every `Problem` can have an arbitrary amount of subproblems in a generic paper exam.

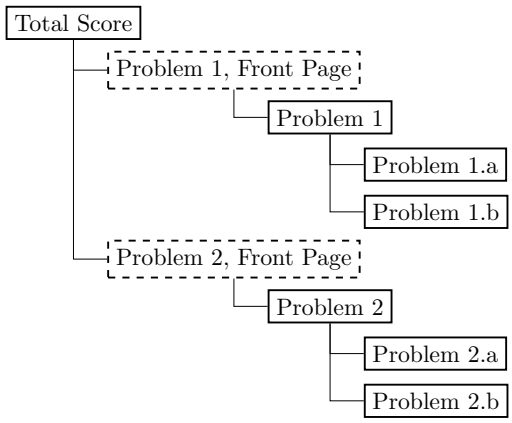


Figure 2: An example tree of Problems.

Dashed lines indicate a score being carried over, not computed.

To iterate through the problems, a depth-first search approach is used [5] as it is similar to the way a paper exam is usually corrected.

3.2.2. View. The graphical interface for the user is kept simple to support fast recognition.

As shown in Figure 3 of the program in execution, a cropped score box and the user input can be seen. The credits of the first correction pass are marked in red, the

doi: 10.2313/NET-2019-10-1_10

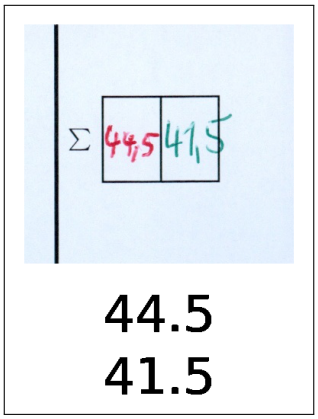


Figure 3: View of the User Interface.

second pass in green. At this stage, the user has entered the values of both scores below the box. After this step the software jumps to the next box immediately.

3.2.3. Controller. The responsibility of the controller is to manage the control flow. The following pseudocode is used to describe its algorithm.

```
load all exam scans into an ExamBatch
for every Exam in the ExamBatch do
  repeat
    display the next Problem (depth-first search)
    start the timer
    await user input
    stop the timer
    store user input and metadata
  until no Problem left in Exam
  store Exam as JSON file [6]
end for
```

Figure 4: The control flow in pseudocode.

As indicated in Figure 4, the controller basically performs a slideshow of score boxes awaiting user input of float values at every step.

3.3. Analysis

To detect miscalculations in the JSON files stored in the recognition phase (Section 3.2), a second python program is used. It is tasked with iterating through the digitized exam data and recompute the scores for every exam. If a mismatch between the calculated and the written credits is detected, an error is recorded. Since this task is significantly less complex than the first program, it is not detailed further.

The main target estimation to be provided by this document is the likelihood p of an exam being falsely graded. Every exam can assume two states, namely being correctly or incorrectly scored. This leads to the assumption of a binomial distribution with parameter p. There are multiple methods for estimating a confidence interval (CI) for a binomial distribution. While the Wald interval method is very prevalent in textbooks, Vollset [7] discourages its use and recommends the Wilson score interval with continuity correction. This method can be applied, because the binomial distribution can be approximated by a normal distribution for large sample sizes.

Let $\hat{p} = \frac{22}{525}$ be the realisation of p in this sample, $n = 525$:

$$\begin{aligned} n\hat{p}(1-\hat{p}) &\geq 9 \\ &\approx 21 \geq 9 \end{aligned} \tag{1}$$

As shown in the Equation (1) our sample is large enough for this continuity correction [8]. The Wilson score interval with continuity correction is determined by [9]:

$$\begin{aligned} L &= \frac{2n\hat{p} + z^2 - 1 - z\sqrt{z^2 - 2 - 1/n + 4\hat{p}(n(1-\hat{p}) + 1)}}{2(n + z^2)} \\ U &= \frac{2n\hat{p} + z^2 + 1 + z\sqrt{z^2 + 2 - 1/n + 4\hat{p}(n(1-\hat{p}) - 1)}}{2(n + z^2)} \end{aligned} \tag{2}$$

Where L is the lower and U the upper bound of the confidence interval. For a confidence level of 95% the value z is the $1 - \frac{1-0.95}{2}$ quantile of the standard normal distribution (Φ is its cumulative distribution function):

$$\begin{aligned} z &= \Phi^{-1}(1 - \frac{1 - 0.95}{2}) \\ &= 1.96 \end{aligned}$$

Equation (2) is later used for the computation of the CI.

In an attempt to interpret the nature of the mistakes made by the correctors, the errors are further dissected. The following attributes of an error are averaged:

1) amount of subproblems that had to be added
2) number of pages the mistake was distributed over
3) absolute offset of the noted score versus the correct one

Finally, the average time needed to recognize a score box or a whole exam is determined. This assesses the effort of a human reading score boxes.

4. Results

The confidence interval of the likelihood of an exam being wrongly corrected is between 2.7% and 6.4% with a mean estimate of 4.2%.

On average, an error is based on 10.5 subproblems that were erroneously added. Furthermore, these values were generally added over 3.9 pages (requiring avg. 1.6 physical page turns). Errors deviate from the correct score by 0.7 points (0.8% of the total score).

All 22 detected errors result from falsely summing subproblem points to a problem, none were made adding the credits on the front page. The score of a Problem was never incorrectly carried over to the front page.

Recognizing a single score box takes about 1.6 seconds, resulting in 85 seconds total per exam. Since a program to facilitate recording of credits is used, these time measures do not include:

1) flipping through the pages
2) localizing score boxes
3) computing the addition

doi: 10.2313/NET-2019-10-1_10

4) fixing own mistakes in this process

Thus the measured time of over 12 hours total is significantly lower than the time required by a human correcting paper exams by hand.

5. Conclusion and future work

Concerning the rate of falsely added exam scores, an interval of 2.7% to 6.4% is high (95% CI). Assuming this value is representative for all university exams and a student participates in four exams on average per semester, from 48% to 80% of bachelor students have at least one of their exam credits miscalculated. Although the impact of 0.8% of the score in these errors seems to be low, exam grading is discrete. This leads to such a deviation having either no effect or result in a significant grade change.

The results of this study indicate that exams containing problems with many subproblems or problems which are distributed over several pages are more prone to error. Further research is needed to validate this claim.

Recognizing all credit values of an exam took 85 seconds, so the total time required for evaluating all exams amounts to over 12 hours. As explained in Section 4, correcting a paper exam without the tools for automation described herein takes significantly longer.

There is a solution for minimizing computational errors and drastically decreasing the required time to evaluate exams. Phillips & Weathers have already pointed out in 1958 that "An alternative would be to have all standardized tests machine-scored" [1]. Automating the recognition and addition of exam scores in the present and future is inevitable and extended research to enhance such software is recommended.

References

[1] B. N. Phillips and G. Weathers, "Analysis of errors made in scoring standardized tests," *Educational and Psychological Measurement*, vol. 18, no. 3, 1958.

[2] G. Carle, "Vorlesung Grundlagen Rechnernetze und Verteilte Systeme," https://www.net.in.tum.de/teaching/ss14/vorlesungen/vorlesung-rechnernetze-und-verteilte-systeme/index.html/, 2014, [Online; accessed 12-June-2019].

[3] Python Software Foundation, "Python language reference, version 3.7," https://docs.python.org/3.7/, 2019, [Online; accessed 17-June-2019].

[4] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, Aug. 1988.

[5] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*. Springer, Oct. 2007. [Online]. Available: https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf

[6] ECMA International, "The json data interchange syntax," http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, Dec. 2017, [Online; accessed 18-June-2019].

[7] S. E. Vollset, "Confidence intervals for a binomial proportion," *Statistics in Medicine*, vol. 12, no. 9, pp. 809–824, 1993.

[8] M. Sachs, *Wahrscheinlichkeitsrechnung und Statistik*. Hanser Fachbuchverlag, Sep. 2003.

[9] R. G. Newcombe, "Two-sided confidence intervals for the single proportion: comparison of seven methods," *Statistics in Medicine*, vol. 17, no. 8, pp. 857–872, 1998.

Network Emulation using Linux Network Namespaces

Daniel Schubert, Benedikt Jaeger*, Max Helm*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga59tek@mytum.de, jaeger@net.in.tum.de, helm@net.in.tum.de

***Abstract*—Testing the behaviour of computer networks can be done with specialized testbeds but they can be expensive and are hard to reconfigure. Therefore other methods like network emulation are used. In an implementation of an emulator, virtual machines can serve as network nodes. However, a more lightweight approach is based on Linux network namespaces. In this paper we describe fundamental emulation features of the Linux operating system that are useful in network emulation. Furthermore, we present Mininet, an emulator that harnesses those features. We examine the API of Mininet and show what happens in the background on a lower level. Finally we present data on the performance of Mininet.**

***Index Terms*—network emulation, virtualization, software-defined networks, OpenFlow**

1. Introduction

Computer networks are already quite complex systems when they contain only a few nodes, with larger network topologies their behaviour gets even more unpredictable and so there is a need for testing. One option is to make use of a testbed consisting of several machines connected to each other but this approach is expensive, inflexible and does not scale very well. Therefore, simulation and emulation frameworks are useful especially in situations where the effects of introducing changes to the network have to be evaluated dynamically. These changes could relate to a protocol, the network architecture or the address scheme. In software defined networks where the functionality of the network can in principle evolve very quickly, such a platform for rapid prototyping can be helpful. Various prototyping environments are already available. In many of those, virtual machines represent nodes which then are connected into a network by virtual interfaces. However there is a different approach that was chosen for the open source network emulator Mininet. There the network is built from processes running in separate Linux network namespaces which are connected by pairs of virtual Ethernet devices. This offers a more lightweight way of network emulation. The remainder of this paper is organized as follows. Section 2 gives a detailed description of network namespaces and other emulation features of the Linux operating system. The Mininet network emulation platform is presented in Section 3, focussing on its API and inner workings. In Section 4 other simulation and emulation tools are discussed. Finally, a conclusion is given in Section 5.

2. Linux virtualization features

In this section we explain the virtualization features of the Linux operating system that are used for the implementation of Mininet.

2.1. Linux Network Namespaces

The concept of namespaces comes in different varieties in the Linux operating system. The common purpose is to offer spaces where processes can be executed in isolation of others regarding various system resources. One of those varieties are network namespaces. They provide processes or groups of processes with their individual network stack including routing tables, network devices, ports and firewall rules among other things. A new network namespace can be created in different ways using the namespace API. One way is to use the clone system call which creates a new process. If the `CLONE_NEWNET` flag is given as an argument the process is provided with a new namespace. Another way is to use the unshare system call from a child process again with the `CLONE_NEWNET` flag to separate it from its parent. At the time of their creation namespaces just contain a private loopback device. Physical network devices can be moved between namespaces but they can always only belong exactly to one of them [1] [2].

2.2. Virtual ethernet devices

In order to enable communication between different network namespaces there exist virtual ethernet devices (veth). They come as pairs and can be thought of as a pipe connecting two namespaces. Using the command shown in Listing 1 a pair of veth interfaces named <name1> and <name2> can be created and the latter is put in the network namespace <netns>. As a result, a connection between the root namespace and the namespace <netns> is established [3].

Listing 1: Shell command to create a virtual ethernet device pair

```
1 ip link add name <name1> type veth
2 peer name <name2> netns <netns>
```

2.3. Control groups

Control groups (cgroups) are to some extent similar to namespaces because they form an environment for

doi: 10.2313/NET-2019-10-1_11

processes with a modified view on system resources. Their purpose is to track and limit the access of processes to resources like cpu time, memory and devices or restrict the number of processes that can be created. Control groups are a hierarchical structure implemented as a pseudo-file-system. To create a new cgroup a folder is added to that file-system. Processes can be assigned to a cgroup by adding their process id to the group's cgroup.procs file. A process can only be part of one group and is automatically removed from any other group on its reassignment. The actual limitation of the resources is carried out by kernel components called controllers or subsystems which are mounted on the file-system. The limits of a cgroup are defined by values written in attribute files of a cgroup folder. [4]

2.4. Traffic control

Apart from controlling the environment a process is running in, it is also possible to influence the network traffic between network namespaces. This is done by influencing the handling of packets at the interfaces of namespaces using Linux traffic control (tc). This way for example the bandwidth of a link can be decreased [5].

3. Mininet

Mininet is an emulator that aims at providing a platform for rapid prototyping of large software defined networks consisting of hundreds of nodes. By using virtualization features on the operating system level it is very lightweight and can therefore be run on commodity hardware. It can be used interactively through a command line interface but there also exists a Python API that allows for the creation of complex network structures by small scripts. In fact almost the complete project itself is written in Python with only some time critical parts implemented in C [6].

3.1. Components

The components emulated by Mininet are hosts, switches, controllers and links. Mininet hosts are simply shell processes that have been given their own network namespaces. Software OpenFlow switches take over the tasks of hardware switches in real networks. By default they run in the root namespace. Typically, controllers are the parts that are tested and therefore beside using emulated ones, it is possible to connect real controllers to the virtualized network. This only requires IP-level connectivity between the controllers and the emulated switches. In order to connect the different nodes of the network virtual links are used. Each link consists of a virtual Ethernet device pair that acts like a tunnel between two virtual interfaces of two different network namespaces [6].

3.2. API

The Miniet API is divided into three levels of abstraction. The low-level API which comprises the base classes for the nodes and links that make up the network, the mid-level API that offers methods that help with the construction of a network as well as with its configuration and eventually the high-level API that provides a class representing a reusable network topology that can be parametrized and instantiated using the command line interface [7].

3.2.1. Low-level API. The most interesting things for us happen at the low-level API because there we can observe the utilization of the operating system's virtualization features. In Listing 2 you can see how the most basic network, containing only two hosts, a connecting switch and a controller can be constructed. The resulting network topology can be seen in Figure 1.

Listing 2: Construction of a simple network using Mininet's low-level API. (Modified from [7])

```
1  h1 = Host('h1')
2  h2 = Host('h2')
3  s1 = OVSSwitch('s1', inNamespace=
       False)
4  c0 = Controller('c0', inNamespace=
       False)
5  Link(h1, s1)
6  Link(h2, s1)
7  h1.setIP('10.1/8')
8  h2.setIP('10.2/8')
9  c0.start()
10 s1.start([ c0 ])
11 print h1.cmd('ping -c1', h2.IP())
```

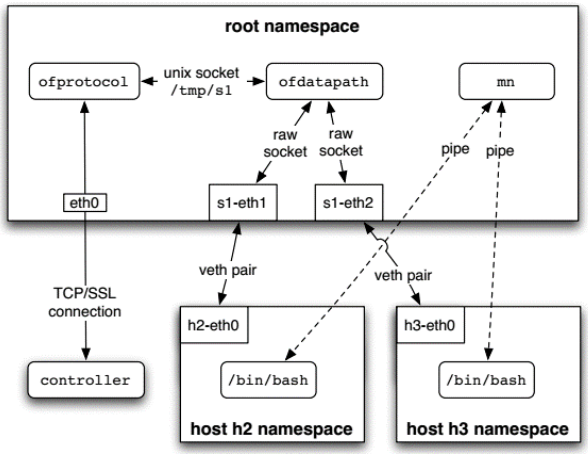


Figure 1: Simple network created by Mininet [6].

At this level, the base classes for the network's components are used directly. In Figure 2 you can see that the classes for three of the four main components of the network e.g. hosts, switches and controllers share a common parent class which is an abstraction for a network node. In the class constructor a new process is spawned that calls a subprogram written in C to create its own new network namespace by using the unshare system call with the `CLONE_NEWNET` flag. The basic Host class which is used in line 1 and 2 in Listing 2 does not differ from its superclass. There are neither new fields nor any new methods. This is different to the CPULimitedHost class where the id of the process started during the instantiation is moved to a cgroup file in order to control the CPU time

doi: 10.2313/NET-2019-10-1_11

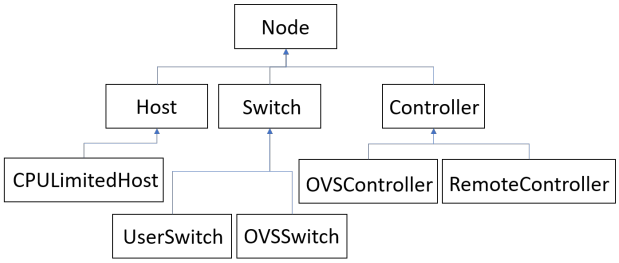


Figure 2: Class diagramm of a part of the Node class hierarchy.

allocated to it. In line 3 an open vSwitch is instantiated that eventually should bridge the two hosts and in line 4 an instance of a controller is created to determine the behaviour of the switch. In order to connect the switch to the hosts two Link objects are used. The basic Link object is just a pair of virtual Ethernet devices which is created as shown in Listing 1. In line 7 and 8 the IP adresses of the default interfaces of the hosts are set to 10.0.0.1 and 10.0.0.2 respectively. The default interface is the one with the lowest port number. In this case it is equal to the virtual Ethernet interfaces that were set up before because the network namespaces of the hosts are created empty except for a loopback interface that is not taken into consideration here. After that the controller's start method is called which brings up an OpenFlow reference controller that listens on port 6653 of the root network namespace. In line 10 an open vSwitch is started using the command shown in Listing 3.

Listing 3: Shell command to start an open vSwitch

```
1  ovs-vsctl add-br <name>
2  -- set bridge <name>
3  controller=[<controllerIds>]
4  -- add-port <name> <intf>
```

Line 1 in Listing 3 shows the basic command to add a bridge called <name>. In line 2-3 the controllers responsible for the switch's behaviour are set and in line 4 an interface is added to the bridge. Line 4 is executed two times in this example adding the two different interfaces that have been created during the instantiation of the two links between the switch and the hosts. Finally, in line 11 of Listing 2 host h2 is pinged from host h1. The cmd method of the Node superclass takes a list of arguments that are combined to a string and run in the shell of the respective node [8].

3.2.2. Mid-level API. If we want to build the same simple network using the mid-level API we can use the code that is shown in Listing 4.

Listing 4: Command to start an open vSwitch

```
1  net = Mininet()
2  h1 = net.addHost('h1')
3  h2 = net.addHost('h2')
4  s1 = net.addSwitch('s1')
5  c0 = net.addController('c0')
6  net.addLink(h1, s1)
7  net.addLink(h2, s1)
8  net.start()
9  print h1.cmd( 'ping -c1', h2.IP())
10 net.stop()
```

When we compare it to the code written using the low-level API we can see some differences. First of all there is a Mininet object which is an abstraction of the network. The network can be started and stopped as a whole as done in line 8 and 10 respectively. Therefore, it is not necessary to start switches or controllers separately. Furthermore, there is no need to manually set IP addresses of interfaces.

3.2.3. High-level API. In the high-level API there is a Topo class that represents a reusable network topology that can be parametrized. This class contains a build method that can be overwritten which orchestrates the creation of a network in essentially the same way as in the mid-level API. A Topo object can be handed to the Mininet constructor as a parameter or can be used as an argument for the command line interface.

3.3. Evaluation of Performance and Accuracy

The inventors of Mininet themselves published a report where they compared the bandwidths that were measured in small networks in the emulator with those of equivalent topologies on a testbed with eight machines. They observed similar TCP results but the results of the emulator were more repeatable and consistent [9].

In a larger study Isaia and Guan [10] examined Mininet with regard to nine different performance categories, including setup time, teardown time, CPU usage, CPU cores load balancing, RAM usage, initial ping delay (IPD), average ping delay, no response failure rate and fair share of resources. Setup and teardown time describe how long it takes to create and destruct a given network topology. To evaluate the CPU usage they took two different measures: the initial CPU usage which is the CPU load after creating the network but before starting to send data across it and the CPU usage during experimentation which is the average CPU load during a specified experiment. For measuring the CPU cores load balancing, an experiment is divided into time intervals. For each of them the standard deviation of core usage is calculated and finally the average is taken. It is a measure of how well the CPU load can be distributed to different cores which is important for scalability. The initial ping delay is the time it takes to ping a node at the start of an experiment. It is significantly larger than at later time points because at the beginning the OpenFlow switches do not contain the necessary flow table rules which first have to be added by a controller. Therefore, in the average ping delay measure the IPD is excluded. No response failure rate is the percentage of unsuccessful ping commands. Fair share of resources in that study was calculated as the coefficient of variation of ping delay between all the hosts when performing a ping command simultaneously. Five different setups with their own network topologies and communication patterns have been tested in 4 different network sizes and each experiment was done 30 times on two different systems that differed in the number of cores, the amount of RAM and the size of the hard disk. The setups covered various

doi: 10.2313/NET-2019-10-1_11

bottlenecks in network communication. The data generated from the experiments showed that the setup time of a network is strongly influenced by the number of switches. In a network of 1000 hosts and one swith the setup time was under ten seconds whereas in a network comprised of two hosts and 1000 switches it reached almost four minutes. But setup time did not always increase linearly with the size of the network and there was no benefit from using the more powerful system. The authors of the study also stated that CPU usage is generally good and that load balancing worked well being positively effected by the number of switches. The initial ping delay is very large compared to the average ping delay and it grows as the number of nodes in the network increases. This is natural because when a connection is used for the first time the forwarding rules have to be added to the switch by a controller. The no response failure rate played a role in larger networks or when the path of a ping packet was long. The fair share of resources measure also gets worse with an increasing number of network nodes. [10]

4. Related work

Apart from Mininet there exist other tools to simulate or emulate computer networks. Simulators try to mimic the behavior of a system model in a more abstract way whereas with emulators the same code can be executed as in the real system. A disadvantage of emulators may be that they run slower than the actual hardware and therefore are not always able to reproduce a realistic timing. In simulators the execution is more flexible and can even be faster then in the real system [11]. On the side of the simulation tools, there is ns-3, an open source discrete-event simulator that was mainly created to support education and research [12]. Others are fs-sdn [13], which as Mininet is targeted at SDN prototyping and the commercial EstiNet X Simulator [14]. An example for network emulation is Mahimahi, a record-and-replay tool that can be used for recording traffic from HTTP-based applications. Later Mahimahi can replay the traffic emulating the network structure that produced it. Linux network spaces are used here as well [15].

Although hundreds of nodes can be emulated on a single machine with Mininet at some point the resources come to an end where it is not possible to make the network larger. To handle this problem Blankstein et al. developed a distributed version of Mininet where the topology of a virtual network is split between multiple computers automatically [16]. This approach has now also been followed by the Mininet community and there is a cluster edition prototype available in the repositiory.

5. Conclusion

In this paper we explained fundamental emulation features of the Linux operating system that are useful in network emulation including network namespaces, virtual Ethernet devices, control groups and traffic control. Furthermore we presented the Mininet emulator, which is based on those features, and described the implementation of its main components. We examined Mininet's API that is divided into three abstraction levels and inspected how it interacts with the operating system to create a simple network. Finally we presented data on performance and accuracy of Mininet and mentioned other tools for network simulation and emulation.

References

[1] "namespaces(7) - linux manual page," http://man7.org/linux/man-pages/man7/namespaces.7.html, [Online; accessed 2019-06-11].

[2] network_namespaces(7) - linux manual page. [Online]. Available: http://man7.org/linux/man-pages/man7/network_namespaces.7.html

[3] veth(4) - linux manual page. [Online]. Available: http://man7.org/linux/man-pages/man4/veth.4.html

[4] "cgroups(7) - linux manual page," http://man7.org/linux/man-pages/man7/cgroups.7.html, [Online; accessed 2019-06-13].

[5] tc(8) - linux man page. [Online]. Available: https://linux.die.net/man/8/tc

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/1868447.1868466

[7] mininet. Introduction to mininet. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#api

[8] ——. Mininet sourcecode. [Online]. Available: https://github.com/mininet/mininet

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Mininet performance fidelity benchmarks," 2012.

[10] P. Isaia and L. Guan, "Performance benchmarking of sdn experimental platforms," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 116–120.

[11] C. Seifert, S. Reißmann, S. Rieger, and C. Pape, "Evaluation von virl, gns3 und mininet als virtual network testbeds in der hochschullehre," in *11. DFN-Forum Kommunikationstechnologien*, P. Müller, B. Neumair, H. Reiser, and G. Dreo Rodosek, Eds. Bonn: Gesellschaft für Informatik e.V., 2018, pp. 103–112.

[12] T. Henderson and M. Lacage. Network simulator 3. [Online]. Available: https://www.nsnam.org

[13] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for sdn prototyping," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 31–36. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/2491185.2491202

[14] EstiNet. Estinet. [Online]. Available: https://www.estinet.com/ns/

[15] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 417–429. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[16] A. Blankstein, S. A. Erickson, and M. Melara, "Mininet clustering," 2013.

What is Deterministic Network Calculus?

Tobias Wasner, Max Helm*, Dominik Scholz*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mail@wasnertobias.me, helm@net.in.tum.de, scholzd@net.in.tum.de

***Abstract*—This paper gives a short introduction to Deterministic Network Calculus. Explains how service guarantees can be provided to error-free packet-switching networks and the limitations of this process. Discusses alternative mathematical approaches, their limitations and corresponding use cases.**

***Index Terms*—deterministic network calculus, worst-case performance, guaranteed service, packet-switching network, traffic shaping, schedulability**

1. Introduction

As shown by Cruz in [1] and [2], Deterministic Network Calculus (DNC) is used to calculate theoretical worst-case performance guarantees for error-free packet-switching networks of queues and schedulers.

DNC, other than Stochastic Network Calculus (SNC), does not use probabilistic terms to characterize performance guarantees. This means that performance guarantees calculated using DNC will hold in any case. This is why DNC is especially useful for hard real-time systems, as those systems assume that missing a single deadline results in a total system failure. [1]

In section 2 the required mathematical background for the application of DNC will be introduced.
In section 3 the application of DNC will be explained.
In section 4 examples for limitations of the application of DNC will be given.
In section 5 alternative frameworks will be briefly introduced and their limitations will be compared to those of DNC.
In section 6 the main results of this paper will be summarized.

2. Background

In order to be able to calculate performance bounds, certain mathematical models have to be applied to the network, which will be described in this section.

2.1. Flows

A comprehensive introduction of the term *flow* is shown by Boudec et al. in [3]. The notation of this section of the paper is derived from Geyer in [4]. The term *flow* is used to describe a unidirectional set of packets which are being sent from a single sender to a single receiver in a packet-switching network [4]. Cumulative arrival functions are used to mathematically model flows. They need to be a member of the following set: [4]

$$F = \{f : \mathbb{R}^+ \to \mathbb{R}^+ \mid \forall 0 \leq t \leq s : f(t) \leq f(s), f(0) = 0\}$$

In the formular s is the time of the end of the flow. This implies that every $A \in F$ is a non-decreasing strictly positive function. A represents the amount of data being sent by the flow in the time interval $[0, t)$.

Furthermore, a flow has a deterministic arrival curve $\alpha \in F$ if its cumulative arrival function A satisfies: [4]

$$\forall 0 \leq s \leq t : A(t) - A(s) \leq \alpha(t - s)$$

In the formular s is the time of the beginning of the constraint and t is the the time of the end of the flow. It is said that the flow A is constrained by α in that case [3]. The cumulative arrival function A can also be used to describe the sending rate as DNC assumes that no packet loss happens.

2.2. Traffic shaping

As shown by Tanenbaum in [5], traffic shaping is needed to constrain flows. Constraining of flows is the basis of all calculations of DNC. Traffic shaping is a technique which smooths out the traffic on the senders' side, rather than on the receivers' side [5]. Two different traffic shaping algorithms will be explained in the following.

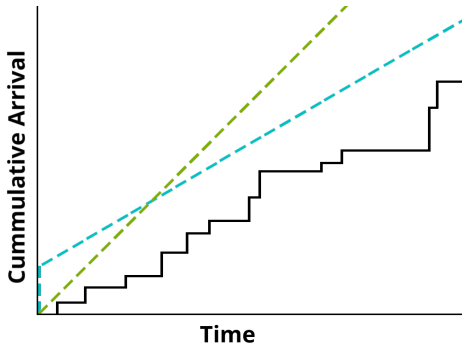


Figure 1: Cumulative byte arrival (solid black line) constrained by a token bucket arrival curve (dashed blue line) and a leaky bucket arrival curve (dashed green line).

doi: 10.2313/NET-2019-10-1_12

2.2.1. Leaky bucket algorithm [5]. This traffic shaping algorithm derives its name from the idea of a bucket being constantly filled with an irregular rate of water (water abstracts packets to be sent over the network) and with one little hole at the bottom of the bucket (packets actually being sent over the network).
The filling of the buffer, therefore, is expressed as the filling of the bucket. The outflow through the little hole is happening with a constant rate r, at least if the bucket is not empty. Once the bucket is fully filled, any further input will simply spill over and is therefore lost.
The rate r can either have the unit number of packets per time frame or data size per time frame. The second approach is useful in the case where not all packets have a fixed data size.

2.2.2. Token bucket algorithm [5]. The content of this bucket are tokens. A token is the allowance to send data in the form of a certain number of packets or bytes. If data has been sent over the network, the number of tokens in the bucket is reduced accordingly. Tokens are added to the bucket at a constant rate r, however, if the bucket is full, no more tokens can be added. If the bucket is empty, data cannot be sent out to the network, it is necessary to wait for tokens to be added in this case.

This algorithm is more flexible in comparison to the leaky bucket algorithm as the output rate is not fixed and the token bucket is fully filled at the time of the beginning of the constraint. Therefore this algorithm allows bigger bursts to happen in comparision with the leaky bucket algorithm. This can also be seen in Figure 1, where different minimum rates r (slopes) are required to constrain the given flow, due to the offset of the token bucket algorithm.

2.3. Servers and service curves

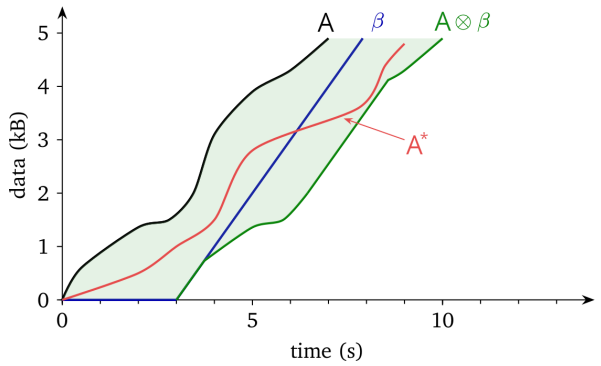


Figure 2: Visualization of the service curve concept [6].

Every single node in a network has a queue and a scheduler, which is an algorithm which decides which packet will be sent next, in case there are multiple packets in the queue waiting to be transferred. The term *server* is used to describe a whole network or certain parts of a network, e.g. a link, a scheduler or a traffic shaper. [4]

Given a deterministic arrival curve A, a server is characterized by its deterministic service curve β, such that the output curve A^* of a flow after traversing the server is defined as: [4]

$$A^*(t) \geq \inf_{t \geq s \geq 0} \{A(s) + \beta(t-s)\} = A \otimes \beta$$

This definition is illustrated by Bemten et al. in [6], correspondingly Figure 2.
As the departure of some data cannot occur before its arrival it is implied that $\forall t \geq 0 : A(t) \geq A^*(t)$.

3. Application

In this section the application of DNC will be explained.

3.1. Service guarantees

DNC provides two different kinds of service guarantees, namely delay bounds and backlog bounds. Both will be introduced in the following.

3.1.1. Delay bounds. The virtual delay d at time t is defined by the following equation: [3]

$$d(t) = \inf_{\tau \geq 0} \{A(t) \leq A^*(t+\tau)\}$$

The term delay bound corresponds to the maximum time that incoming data has to wait before being processed by the server. In mathematical terms this can be expressed using the following equation: [4]

$$A^*(t) - A(t-s) \leq \sup_{t \geq 0} \{d(t)\}$$

3.1.2. Backlog bounds. The backlog b at time t is defined by the following equation: [3]

$$b(t) = A(t) - A^*(t)$$

The term backlog bound corresponds to the maximum amount of data that will have to wait before being processed by the server. In mathematical terms this can be expressed using the following equation: [4]

$$A(t) - A^*(t) \leq \sup_{t \geq 0} \{b(t)\}$$

3.2. Generalized Processor Sharing

Generalized Processor Sharing (GPS) is the ideal form of per flow queuing. Per flow queuing provides isolation of flows and therefore service guarantees differentiated per flow. Numerous practical implementations of GPS have been proposed in the literature. Each differs in their provided service guarantees and their implementation complexity. Practical implementations of GPS will be introduced in the following. [3]

3.2.1. Practical Generalized Processor Sharing. Practical Generalized Processor Sharing (PGPS) implements GPS using one First In First Out (FIFO) queue per flow [3]. Each queue is assigned a priority [4]. Based on the assigned priority the available bandwidth is shared accordingly [4].

doi: 10.2313/NET-2019-10-1_12

3.2.2. Guaranteed Rate Schedulers. All practical implementations of GPS fit in the framework Guaranteed Rate Schedulers (GRS). This approach considers a server with FIFO-scheduling and a constant bit rate r. [3]
Furthermore, T_i is defined as the arrival time, T_i' as the departure time and l_i as the length in bits of the ith packet, ordered by arrival time [3].
Assuming $T_1 \geq 0$, where T_1 is the arrival time of the packet which arrived earliest, the definition of T_i' is the following: [3]

$$T_i' = \begin{cases} 0 & \text{if } i = 0 \\ max\{T_i, T_{i-1}'\} + \frac{l_i}{r} & \text{if i > 0} \end{cases}$$

This means that packet i starts its service at $max\{T_i, T_{i-1}'\}$ and ends at $max\{T_i, T_{i-1}'\} + \frac{l_i}{r}$ [3].

3.3. Schedulability

Networks are not usually built in the way that a single piece of hardware is being exclusively used by one single flow. Quite the opposite, it is extremely common that a single node in a network has to handle transfers for multiple flows in parallel. Schedulability enables that service guarantees can still be made in that case. This is done in the following way: [3]

When a node is affected by a new flow it has to reserve two kinds of resources locally: bandwidth and buffer size. In order to be able to reserve those resources the quantity has to be determined. To calculate the needed bandwidth and buffer size we have to keep the service curve and the arrival curve constraints of the flow in mind. The most general framework which is making those calculations possible is named Service Curve Earliest Deadline First. [3]

3.3.1. Earliest Deadline First. The concept of Earliest Deadline First (EDF) schedulers assumes that there is a list for every corresponding flow which contains the arrived packets which are waiting to be transferred further. Furthermore, a deadline D_i^n is allocated to every nth packet of every ith flow. [3]
At every time slot the scheduler picks one packet with the earliest deadline out of all packets independent of the flow. This general concept contains no further assumption of how the deadline is mathematically allocated. For that reason multiple concepts of other scheduler types - e.g. FIFO - can be fit in this concept as well. [3]

3.3.2. Service Curve Earliest Deadline First Schedulers. As shown in subsection 3.3.1 EDF schedulers assume that there is a deadline allocated to every packet. Service Curve Earliest Deadline First (SCEDF) schedulers allocate the deadlines for every packet in that way that every ith flow does have β_i as a service curve. [3]

3.4. Time Analysis Methods

It is explained in section 3.1 how service guarantees can be calculated for individual servers. The Time Analysis Model enables to calculate delay bounds for flows which traverse multiple servers.

3.4.1. Total Flow Analysis. As shown by Heidinger in [7], this method of calculating delay bounds is done in the following way: The delay bounds are calculated per traversing node and then added up. For that reason this method is also known as *node-by-node analysis*.
The problem with this method is that the delay bounds will be calculated overly pessimistic, because bursts are not only paid at the first traversing node, but at every traversing node. [7]

3.4.2. Separate Flow Analysis. This method of calculating delay bounds is done in the following way: The service curves are calculated per traversing edge, added up and then the horizontal deviation is used as a delay bound. [7]
In that way bursts will be only paid at the first traversing node. The problem with this method is that the delay bounds will be calculated overly pessimistic, if a flow is multiplexed several times. [7]

3.4.3. Pay Multiplexing Only Once. As shown by Schmitt et al. in [8], with this method overly pessimistic calculations of delay bounds do not happen, if a flow is multiplexed several times. This method is based on separate flow analysis as shown in section 3.4.2.

4. Limitations

In this section the limitations of the application of DNC will be explained.

4.1. Cyclic dependencies

As shown by Schiøler et al. in [9], cyclic dependencies in dataflows can not be modeled using DNC without any modifications. This problem is still under active research and there is already an approach named Cyclic Network Calculus (CyNC) which aims to support that use case, but has not yet produced correct results in every case [9].
CyNC is based on DNC but extends the theoretical basis. Several modifications of DNC are already proposed and even more are needed in the future to fully support this application area. [9]

4.2. Feedback loops

One might think that protocols which include feedback loops - e.g. TCP - cannot be modeled using DNC. In fact, this assumption has been proven wrong, as Baccelli et al. have shown in [10] that TCP can be modeled using max-plus algebra. However this is not a common use case for DNC, because the used traffic shaping in DNC is about regulating the average rate and burstiness of data transmission whereas the sliding window protocols, such as TCP, only limit the amount of data in transit at once [5].

4.3. Overprovisioning

DNC aims to determine the actual worst case which can be seen in Figure 3. Practically the calculation of overly pessimistic upper bounds can be observed frequently using DNC as shown by Fidler in [11]. However,

doi: 10.2313/NET-2019-10-1_12

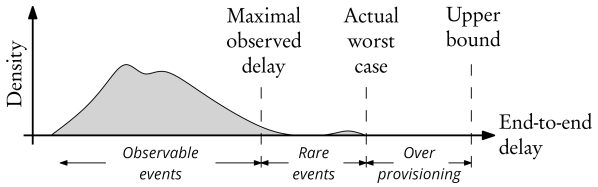


Figure 3: Visualization and naming of delay bounds. [4]

determining the exact actual worst-case is NP-hard [4].
For systems which are not of the type hard real-time - e.g. best-effort or soft real-time systems - those overly pessimistic calculations of the actual worst case can easily make the calculations of DNC worthless. Even if the actual worst case is not calculated overly pessimistic it still may be attained rarely and may not even have the effect of breaking the whole system as silently assumed by DNC. [11]

5. Alternative approaches

In this section alternative approaches to calculate performance guarantees will be briefly introduced and their limitations will be compared to those of DNC.
Figure 4 provides a broad overview of the different use cases of several alternative approaches. Some approaches will be explained further in the following subsections.

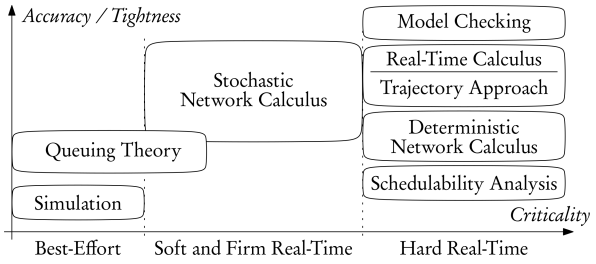


Figure 4: An overview of the use cases of several alternative approaches [4].

5.1. Stochastic Network Calculus

As described in section 4.3, DNC is not designed to model systems which are not of the hard real-time type. SNC characterizes performance bounds in probabilistic terms [1], which makes it more suitable to model firm or soft real-time systems [4].

5.2. Queuing theory

As shown by Lipsky in [12], this approach uses probabilistic terms to characterize performance bounds, as well as SNC. Furthermore, it does not analyze the worst-case, as DNC or SNC does, but the average-case [12], which makes it more suitable to model best-effort systems [4].

6. Conclusion and future work

In this paper we have shown how Deterministic Network Calculus can be used to calculate theoretical worst-case performance guarantees for error-free packet-switching networks of queues and schedulers.

In section 2, we started to define the term flow, which is used to describe a unidirectional set of packets which are being sent from a single sender to a single receiver. We showed what it means that a flow is constrained and how constrains of flows can be calculated.
In section 3, the concept of servers and service curves, delay bounds and backlog bounds have been defined and explained. We put all concepts together and showed how service guarantees can be calculated even when multiple traversed servers and active flows are involved.
In section 4, we have shown what the limitations of the application of DNC are.
In section 5, we briefly introduced alternative approaches to calculate performance bounds and compare their limitations and corresponding use cases.

Overall we have seen that the analysis of the network is bound to the exact requirements of each individual flow. Therefore the exact requirements have to be known in beforehand to be able to calculate performance bounds using DNC. Therefore the constructed network is bound to the initial planned use case, also in terms of hardware. That fact makes the use cases of DNC inflexible. The effort of defining those exact requirements and making the calculations is only worth it in special real-world applications.
On one hand, the calculated performance guarantees of DNC can be valuable whenever the criticality is high, e.g. when even lives are at stake. On the other hand, the calculations of DNC assume that the network is error-free, which means that a single point of failure could break calculated performance guarantees partly or even fully. This problem still has to be taken into mind. One real-world application is the validation of embedded networks inside the Airbus A380 and A350 [4].

References

[1] R. L. Cruz, "A calculus for network delay. i. network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, Jan 1991.

[2] ——, "A calculus for network delay. ii. network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, Jan 1991.

[3] J. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003.

[4] F. Geyer, "Quality-of-service and network calculus," 2019, [accessed 20-May-2019]. [Online]. Available: https://acn.net.in.tum.de/slides/190205_chap12_QoS_Network_Calculus.pdf

[5] A. Tanenbaum, *Computer Networks*, ser. Computer Networks. Prentice Hall PTR, 2003, no. S. 3.

[6] A. V. Bemten and W. Kellerer, "Network calculus: A comprehensive guide," 2016, [accessed 23-June-2019]. [Online]. Available: https://mediatum.ub.tum.de/doc/1328613/1328613.pdf

[7] E. Heidinger, "Worst case analysis - network calculus," 2012, [accessed 20-May-2019]. [Online]. Available: https://www.net.in.tum.de/pub/systemperformanz/ss2012/skript/networkcalculus.pdf

[8] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic, "Improving performance bounds in feed-forward networks by paying multiplexing only once," in *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, March 2008, pp. 1–15.

doi: 10.2313/NET-2019-10-1_12

[9] H. Schiøler, J. J. Jessen, J. D. Nielsen, and K. G. Larsen, "Network calculus for real time analysis of embedded systems with cyclic task dependencies," in *Computers and Their Applications*. Citeseer, 2005, pp. 326–332.

[10] F. Baccelli and D. Hong, "Tcp is max-plus linear and what it tells us on its throughput," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 219–230, Aug. 2000, [accessed 23-May-2019]. [Online]. Available: https://doi.acm.org/10.1145/347057.347548

[11] M. Fidler, "Survey of deterministic and stochastic service curve models in the network calculus," *IEEE Communications Surveys Tutorials*, vol. 12, no. 1, pp. 59–86, First 2010.

[12] L. Lipsky, *Queueing Theory - A Linear Algebraic Approach*, 2nd ed. Berlin Heidelberg: Springer Science & Business Media, 2008.

Optimization of Decision Trees for TCP Performance Root Cause Analysis

Marco Weiss, Simon Bauer*, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: marco.weiss@tum.de, bauersi@net.in.tum.de, jaeger@net.in.tum.de

***Abstract*—Identifying the root cause for TCP throughput limitations helps to improve network performance and user experience. In previous work, decision trees (DTs) have been used as a tool for TCP performance root cause analysis (RCA) based on passive network measurements. The topology of those trees has been designed based on the working principles of TCP and the decision thresholds were chosen by inspection of measured data. We present how genetic algorithms (GAs) can be used to further optimize those DTs by fitting their threshold values to a dataset of synthetic network traffic with known root causes. In the next step, machine learning algorithms, namely decision tree learning, random forest and extremely randomized trees, are used to build DT-based classifiers in a purely data-driven fashion. It is shown that the classification accuracy of the hand-crafted DTs could be improved after optimizing their thresholds with our GA approach. However, even the optimized hand-crafted DTs were outperformed by the machine learning approaches with a significant margin.**

***Index Terms*—TCP IP, decision trees, machine learning, evolutionary computation**

1. Introduction

As TCP is one of the most widely used transport layer protocols, any TCP performance issues might directly impact its users. RCA tools help to identify and overcome such issues. In [1] and [2], Siekkinen et al. and Stemplinger developed tools for TCP performance RCA based on passive network measurements using DTs. DTs are an intuitive and interpretable technique that employs the divide-and-conquer strategy for decision making. Due to their intuitive use, it is possible to design DTs from hand by analyzing the functionality of the underlying system. This is especially the case for "white-box" systems like TCP, where all internal structures and functions are in principal known. A different approach for building DTs comes from the field of machine learning, where the tree structure and its decision rules are purely based on statistical properties of data generated by the system. Despite having no knowledge of the data generating process, DT learning algorithms perform quite well in practice.

In this work, we aim to evaluate both approaches on the same dataset. To this end, we do not only use the dataset to train classifiers with different DT learning algorithms, but we also try to further improve the classification performance of the existing hand-crafted DTs by fitting their threshold values to our data. This is in fact not trivial because common DT learning algorithms need to have control over both the tree topology and the decision thresholds to achieve good performance. Thus, we need to formulate the task as a general optimization problem. To solve it, we chose to use GAs for two main reasons: GAs are easy to implement and they impose almost no limitations to the optimization problem at hand, compared to e.g. gradient-based methods that require a differentiable objective function or linear programming that requires a linear objective function (both is not the case for DT optimization which is in fact NP-complete [3]).

The remainder of this paper is organized as follows: First, related work to the fields of TCP RCA, DTs and GAs for DT oprimization is presented in section 2. After introducing our dataset in section 3, we present the baseline DTs, how they can be optimized with GAs and different machine learning approaches in section 4. The setup and results from our experiments are presented in section 5 before summarizing our findings in section 6.

2. Related work

In [4], Zhang et al. were the first to perform a holistic analysis on the limiting factors of throughput in internet connections. Based on their findings, they developed T-RAT, a tool for RCA based on trace files. Siekkinen et al. extend this work in [1] to overcome limitations of T-RAT that are discussed in [5] in detail. They introduce a set of quantitative metric, called *limitation scores*, which can be inferred from TCP headers and are then used in a DT-based RCA tool. In [2], Stemplinger extends their approach and uses synthetic training data generated by the Mininet network emulator to adapt the decision thresholds to more recent congestion control algorithms.

Closely related to the work done on throughput RCA is [6], where Jaiswal et al. aim to estimate the sender's congestion window size and the connection round trip time (RTT) from passive measurements. They explicitly demarcate their work from [4], but claim that the limitating factors of a TCP connection can be determined based on congestion window size and RTT. This work is of particular interest because in [7] and [8], Hagos et al. use machine learning techniques, namely random forest, gradient boosting and recurrent neural networks, to significantly improve prediction performance compared to the state machine approach from [6]. Quite similar to the machine learning part of our work is [9], where El Khayat et al. use decision tree boosting to discriminate between TCP package loss due to overflow or link errors in wireless networks.

doi: 10.2313/NET-2019-10-1_13

The foundations of DTs, their extensions and learning algorithms can be found in [10], [11]. Decision tree learning, i.e. finding the combination of optimal split dimensions and thresholds, has been proven to be NP-complete [3]. State-of-the-art decision tree learning algorithms, e.g. classification and regression tree (CART) as implemented in the scikit-learn machine learning library [12], use a greedy heuristic to determine the split that maximizes the purity of the resulting distributions or the accuracy for every new node. In general, DTs have several advantages as they are easy to interpret, robust to outliers and scale well to large datasets. However, they are considered high-variance estimators, meaning their prediction performance might be worse than with other machine learning methods in some cases. To deal with this issue, several extensions to DTs have been proposed. The probably most-widely known one is random forests by Breiman [13], where the prediction is computed as the average of an ensemble of different DTs. Building on that, Geurts et al. later introduced extremely randomized trees (extra-trees), where the construction of all trees in the ensemble is completely randomized instead of using a split heuristic [14]. We will refer to both techniques as ensemble methods in the following.

A fundamental introduction to genetic algorithms is given in [15]. In combination with DTs, GAs have previously been used for pre-processing, i.e. selecting the best subset of a large feature space which is then used as input for a heuristic-based decision tree learning algorithm [16]. There exist also attempts to directly use GAs to build DTs. In [17], Papagelis et al. achieved comparable classification performance to heuristic-based approaches when optimizing their DT with GAs. In [18], Cha et al. used GA-based optimization to build compact, nearly-optimal decision trees.

3. RCA Dataset

We train and evaluate all our models on the dataset from [2]. It was generated using the network emulator Mininet with different test setups and network topologies to enforce different throughput limitations. In the context of TCP performance RCA, those throughput limitations will be referred to as the *root causes*. As in [1] and [2], only bulk transfer periods (BTP), i.e. the time window in which throughput is limited by the network connection and not the sending application, are analyzed. After the BTPs have been isolated from the application limited periods, five *limitation scores* were calculated for each BTP. All limitation scores are based on information contained in the TCP headers, so measurements can be obtained passively at any point in the connection [1]. In the following, we will give a brief summary of the possible root causes and limitation scores derived in [1], [2] and provide an overview of the used dataset.

3.1. Root Causes

Capacity bottleneck: The throughput of a connection can be limited by the bandwidth available at the bottleneck link. We distinguish between unshared bottlenecks (ub), where our connection uses the entire bandwidth of the bottleneck link and shared bottlenecks (sb), where parts of the bottleneck bandwidth are used for other transmissions.

Receiver window (rw): The receiver-side application sets the size of the receiver window, i.e. the number of possible bytes per packet, based on how fast it can process incoming data. The receiver window can be static or dynamically scaled by the receiver application during transmission. In the first case, a combination of small default window size, high bandwidth and rather long transmission times can limit the throughput unintentionally. In the latter case, the application can limit throughput intentionally if it cannot process incoming data fast enough.

Congestion avoidance (cw): On sender-side, the congestion control algorithm tries to estimate the best sending rate for the connection. Depending on its implementation, there might be phases in which the throughput is solely limited by the congestion control algorithm, e.g. at the beginning of a connection or after the congestion window was lowered due to detected packet loss.

3.2. Limitation Scores

Dispersion score: The dispersion score is defined as

$$s_{disp} = 1 - \frac{TP}{C}, \tag{1}$$

where TP is the average throughput of the BTP and C is the capacity of the bottleneck link. The dispersion score can be used to determine whether a connection is limited by an unshared bottleneck ($s_{disp} \approx 0$) or a shared bottleneck ($s_{disp} > 0$).

Retransmission score: The retransmission score is defined as the ratio of retransmitted bytes to transmitted bytes

$$s_{retr} = \frac{n_{retr}}{n_{trans}}. \tag{2}$$

A high retransmission score is an indicator for a network bottleneck where the link buffer is filled up until packets are dropped and have to be retransmitted.

RTT score: The RTT score is an alternative to the retransmission score for detecting network bottlenecks and is defined as

$$s_{RTT} = \frac{avg(RTT)}{min(RTT)}, \tag{3}$$

where RTT is the round trip time measurement of a packet, i.e. the time between sending the packet and receiving the acknowledgement. Essentially, the RTT score indicates a bottleneck if the current RTT is higher than the lowest possible RTT measured so far.

Receiver window score: The receiver window score s_{rwnd} quantifies how much the sender is limited by the advertised receiver window. This is done by comparing the number of outstanding bytes and the receiver advertised window size over time. If the difference between both values is small, i.e. below a certain threshold, the congestion window of the sender is close to the limit set by the receiver. The receiver window score is then calculated as the average number of occurrences for this event over the duration of a BTP.

Burstiness score: The burstiness score (or b-score) s_b can be used to determine whether connections with high receiver window score are actually limited by the size of

doi: 10.2313/NET-2019-10-1_13

the receiver window or by a bandwidth bottleneck. When transmitting n packages in a receiver-limited setting, a burst of $n-1$ short inter-arrival times (IAT) is followed by a long IAT because the sender has to wait for the receiver's acknowledgement. If the transmission is however bandwidth-limited, the distribution of the IATs is more even due to buffering at the bottleneck link.

3.3. Overview

The complete dataset consists of 533 measurements that were taken during different BTPs. Each measurement vector $x = (s_{disp}, s_{retr}, s_{RTT}, s_{rwnd}, s_b) \in \mathbb{R}^5$ is labelled with the true root cause $y \in \{cw, rw, sb, ub\}$. It has to be noted that different combinations of root causes are possible in theory. In practice however, there is usually a single dominant root cause per connection [1]. To visualize the dataset, principle component analysis (PCA) is applied to the normalized data. Using the first two components, the dataset can be plotted as shown in Figure 1. It has to be noted that the visualization captures only 66% of the variance in the data. Nevertheless, this still gives a first impression how the data is structured. It can e.g. already be seen that the all measurements with $y = rw$ can be linearly separated from the other classes.

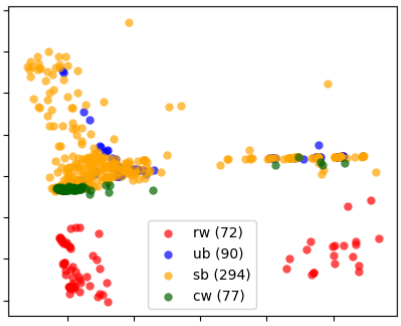


Figure 1: Dataset visualization using the first two components from PCA (capturing 66% of the variance in the data). The number in brackets indicates the number of data points per class.

4. Methodology

In the following, different strategies will be described to derive DTs for TCP performance RCA from the available data. As baseline, we use two hand-crafted DTs from [1] and [2], where decision thresholds were manually defined in [2] based on inspection of the dataset. In the next step, we aim to improve the classification accuracy of those trees by using GA to optimize the thresholds. To compare the performance of hand-crafted DTs with a purely data-driven machine learning approach, we use DT learning, namely CART, in chapter 4.3. In chapter 4.4, we use ensemble methods, namely random forest and extra-trees, which are expected to improve classification performance compared to single DT learning.

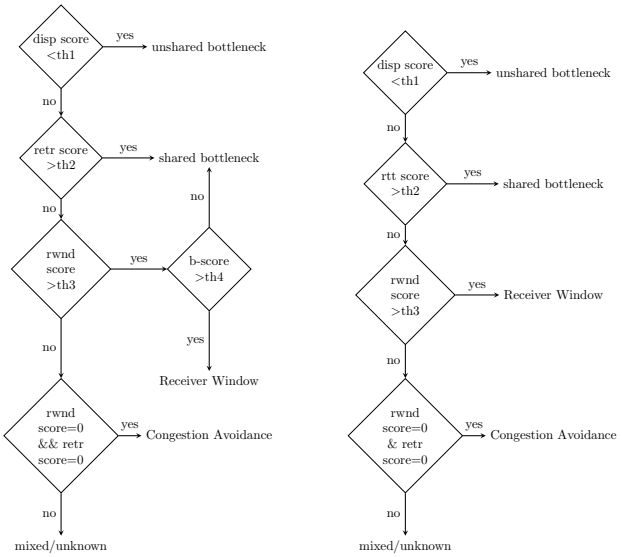


Figure 2: Baseline decision tree (a) and baseline RTT decision tree (b), both taken from [2].

4.1. Fitting by Inspection

In [1], Siekkinen et al. proposed a DT based on dispersion score, retransmission score, receiver window score and burstiness score. In [2], the threshold values of this DT where refined by manually analyzing the dataset described in the previous chapter. In the following, we will refer to this as the *baseline* approach. Additionally, a modified version of the baseline tree was presented using the RTT score instead of retransmission and burstiness score, called *baseline RTT*. The trees are depicted in Figure 2. It has to be pointed out that threshold fitting of both trees was done using the complete dataset. For our other approaches below, we perform a 80/20 train-test split to measure generalization performance on unseen data.

4.2. Optimization with Genetic Algorithm

The general idea of GAs is that evolution in biology can be seen as an optimization problem: In a population of individuals, the ones adapted best to their environment survive the longest and the older an individual gets the more time it has to reproduce. Thus, the genes of the best-fitting individuals spread while "unsuitable" genes vanish over many generations. In a more mathematical sense, natural selection can be seen as a search heuristic to find the best genes (hence the name Genetic Algorithms). To apply this heuristic, candidate solutions to the optimization problem have to be encoded in chromosomes. Their fitness is evaluated based on an objective function and the fittest candidate solutions generate new candidate solutions based on genetic operators [15]. In the following, we will describe how this process can be applied to the decision threshold optimization problem. The resulting DTs will be referred to as *optimized* and *optimized RTT*.

The first step is to define the encoding of candidate solutions. Compared to other approaches e.g. in [17] where a candidate solution has to encode a complete tree topology,

doi: 10.2313/NET-2019-10-1_13

our encoding is rather simple because we only want to optimize the decision thresholds for a fixed topology. In accordance with the notation of Figure 2, we have candidate solutions in the form of $c_{base} = \{th_1, th_2, th_3, th_4\}$ and $c_{baseRTT} = \{th_1, th_2, th_3\}$, where $\forall j \ th_j \in \mathcal{T}_j$ and $\mathcal{T}_j$ denotes the set of possible threshold values for the j-th dimension of the input vector. Using a set of discrete threshold values instead of real-valued numbers drastically reduces the search space without affecting the training accuracy of the candidate solutions. Analogously to heuristic-based DT learning [11], the threshold sets for a given training dataset $\mathcal{D}$ are obtained by $\mathcal{T}_j = \{x_j : x \in \mathcal{D}\}$.

After the encoding has been defined, a start population is created by generating n_{pop} individuals with random values from the threshold sets. To evaluate the *fitness* of a candidate solution, accuracy on the training dataset is used. *Selection* is done in an elitist way: The best n_{elit} individuals are kept for the next generation without any changes. On the remaining $n_{pop} - n_{elit}$ individuals, *crossover* and *mutation* can be applied. In original GA implementations, one or more crossover points are chosen at which the parent chromosomes are split and exchanged between both partners to maintain so-called building blocks [15]. In our implementation however, the order of the elements is completely arbitrary and not meaningful, so we do not need to maintain any building block structures in the solution. Therefore, we choose parameterized uniform crossover with probability of 50%, which essentially means that the elements in the offspring chromosome are randomly chosen from both parents. For every offspring, the event mutation happens with a configurable probability. In case of mutation, one element c_j of the chromosome is replaced by a random element in $\mathcal{T}_j$. In nature, mutation can only happen during reproduction. This does of course not apply to a virtual implementation of such genetic operators, so it is possible to apply mutation to any individual and not only to new offsprings. In Figure 3, graphical examples for the genetic operators described above are given.

Starting with the initial population, optimization is an iterative process where every iteration corresponds to a new *generation*. After a fixed number of generations, i.e. when convergence is expected based on experiments presented in the next section, the best candidate solution of the final generation is returned. To avoid converging to a local minimum, GAs try to maintain a set of good, but possibly very different solutions. Random mutation also helps solution candidates to overcome local minima. The effectiveness of those concepts highly relies on a suitable combination of hyperparameters, primarily the population size and the probabilities for crossover and mutation. We tune those hyperparameters by comparing the results from multiple runs of GA optimization to the actual global optimum obtained from brute force calculations. Detailed setup and results of those experiments can be found in section 5.1.

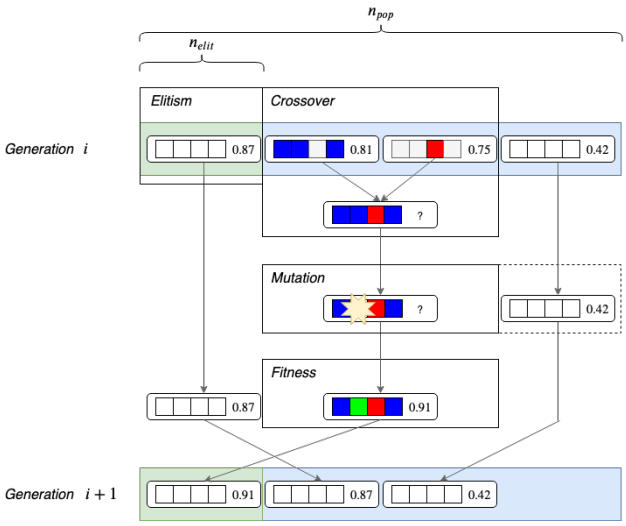


Figure 3: Schematic visualization of genetic operators. White boxes with rounded corners represent candidate solutions with four genes (squares on the left) and a fitness score (right). Red and blue color indicate the origin of a gene during crossover. After mutation, one gene in the offspring is replaced by a random value from the threshold set (green color). The dashed line indicates that depending on the implementation, mutation is also possible for individuals that are not the result of a crossover operator. For clarity, only a single crossover operation is shown and only one individual is passed to the next generation. In practice, both has to be done multiple times to keep the population size constant.

4.3. Decision Tree Learning Algorithm

For DT learning, we use the DT classifier class from scikit-learn [12]. It uses the CART algorithm and the best split is chosen based on the Gini score, a measurement for the impurity of a distribution. As pointed out in the scikit-learn documentation, the training algorithm is biased towards the dominant classes when training a DT on an unbalanced dataset, i.e. data with an uneven distribution of class labels. To overcome this issue and still make use of the complete dataset, each class is weighted with the reciprocal of its relative occurrence. Before training the tree on the complete training dataset, we perform grid search with K-fold cross validation to find the set of hyperparameters that performs best on the test data. A good selection of hyperparameters is mainly important to prevent overfitting. Without any limitations, the tree can be grown until every leaf node is pure, thus giving a training accuracy of 100 % but bad generalization performance. We tune the following hyperparameters to control growth of the tree: Maximum depth, maximum number of leaf nodes, minimum impurity decrease and minimum number of samples for a split.

It has to be noted that in contrast to our baseline trees, the scikit-learn DT implementation does not output *mixed/unknown* as possible root cause. The classifiers always predicts the most likely class, i.e. the one dominating the distribution in the leaf node - a class which is not in the training dataset cannot be predicted. If a classification as *mixed/unknown* is desired, a discrimination based on the likelihood of the predicted class could be a possible solution.

doi: 10.2313/NET-2019-10-1_13

4.4. Ensemble Methods

The scikit-learn library [12] also provides implementations of random forest and extra-trees that we use here. The most important hyperparameters for both methods are the number of trees in the ensemble and the number of features that are considered for every split. For every tree that is built within the ensemble, the same hyperparameters for DT learning apply as discussed in the previous section. In contrast to a single DT however, it is recommended to grow all trees to full size because generalization is essentially achieved by averaging over all trees in the ensemble.

5. Experiments

In the following, two different experiments are presented. First, it is shown on small subsets of the dataset that our GA optimization converges to a nearly-optimal solution with a high probability using a suitable set of hyperparameters. In the second experiment, DT learning and ensemble methods are trained on the dataset and compared to the accuracy of the hand-crafted DTs.

5.1. Convergence of Genetic Algorithm

As motivated in section 4.2, we want to obtain a good set of hyperparameters for our GA-based DT optimization. To this end, we create 3 subsets by randomly sampling 10% of the training dataset. We use a brute-force approach to obtain the parameter sets of the baseline and the baseline RTT tree that maximize the training accuracy on every subset. The resulting accuracies, marked as horizontal dashed lines in Figure 4, are then used as benchmark for the GA-based optimization. To account for the reduced search space by using only 10% of the data, we scale down the population size to 30. We run the GA 10 times per subset and per DT with different random seeds and show the average best-of-generation fitness in Figure 4. It can be seen that the GA converges to a nearly-optimal solution with high probability, while its average training time is faster than the brute-force approach by approximately factor 100. Due to the reduced number of threshold parameters, optimization of the baseline RTT tree with 3 parameters converges in fewer generations than the baseline tree with 4 parameters. Based on tuning the GA hyperparameters to good convergence, we use a crossover probability of 0.5, mutation probability of 0.2 and an elitism ratio of 0.1. We found that a population size of 100 is good compromise between final accuracy and training time on the complete dataset.

5.2. Comparison to Decision Tree Learning Algorithm and Ensemble Methods

As described in section 4.3, we perform grid-search and K-fold cross validation to determine the best hyperparameter set for the DT learning algorithm. As it can be seen in Figure 5, overfitting is in fact not a problem in our case. Although training accuracy is at 100%, there is no significant decrease of validation accuracy. This could be an indicator that there might not be significant noise

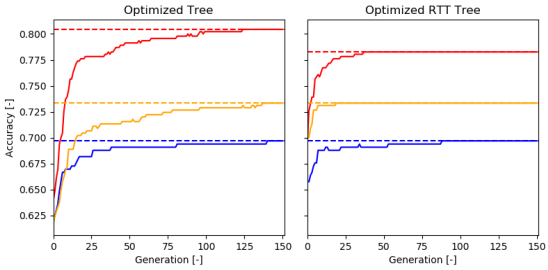


Figure 4: Best-in-generation accuracy of DTs optimized with GA compared to upper boundary (dashed line) for 3 small training data subsets.

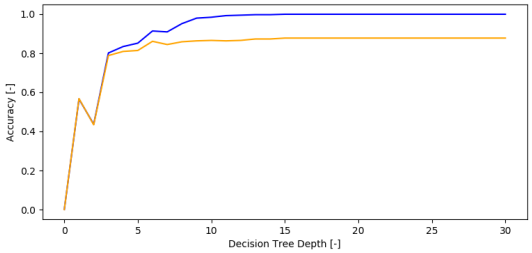


Figure 5: Training accuracy (blue) and validation accuracy (orange) of the DT learning algorithm as function of maximum depth.

in the synthetic training data. However, a more detailed analysis of this phenomena is required.

For our final evaluation, we use a maximum depth of 15. It has to be noted that a smaller tree, e.g. with depth of 5, achieves only slightly worse performance. In Figure 6, the resulting DT is shown up to a depth of 2 for the sake of readability. It can be seen that the first split separates all points of class *rw* in the training set from the other classes as expected from Figure 1. This leads to a missclassification rate of 0% for class *rw* on the training data. The hand-crafted DTs, which have a missclassification rate of over 34% for this class [2], could maybe be improved by also performing the split on s_{rwnd} first.

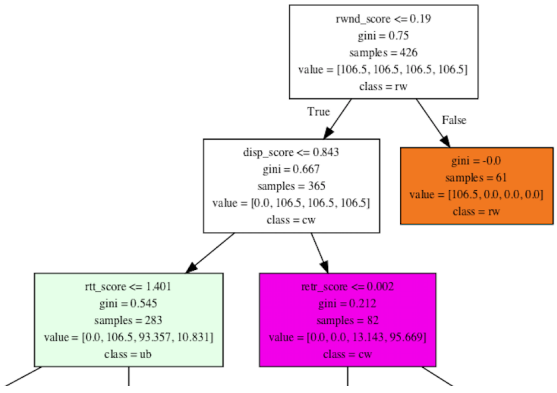


Figure 6: DT fitted to the training data shown up to a depth of 2.

Training of both ensemble methods as implemented in the scikit-learn library is comparable straight forward using the default hyperparameters. The final accuracies of

doi: 10.2313/NET-2019-10-1_13

all presented approaches are listed in Table 1. It has to be noted that the accuracy of both baseline trees is somewhat biased. On the one hand, the decision thresholds were determined using the complete dataset, whereas all other approaches are tested on unseen data. On the other hand, the baseline trees and their GA-optimized versions output classifications of type *mixed/unknown*, which might be useful in some cases in practice, but is always considered a wrong prediction when calculating accuracy.

Method	Train Time	Accuracy	Improvement
Baseline	-	0.73	-
Baseline RTT	-	0.70	-
Optimized	46.2s	0.79	8.2%[1]
Optimized RTT	44.2s	0.75	7.1%[2]
DT learning	< 0.1s	0.92	26.0%[1]
Random Forest	0.1s	0.93	27.4%[1]
Extra-Trees	0.1s	0.94	28.8%[1]

TABLE 1: Train times on 1.6 GHz Intel Core i5, final accuracies on test data and relative improvement compared to the respective baseline tree accuracy.

[1] Compared to *baseline* tree.
[2] Compared to *baseline RTT* tree.

6. Conclusion and Future Work

The main goal of this work was to optimize existing decision trees for TCP performance RCA on a given dataset. With GA-based optimization, we were able to improve their classification accuracy by up to 8%. We could show for small subsets of the data that the GA-based optimization of DTs converges to a near-optimal solution with high probability. It can therefore be assumed that the performance of the baseline DTs is limited by their design. Consequently, we used DT learning to optimize not only the decision thresholds of the DT but also its topology. By doing so, we could improve classification performance by 26% compared to the baseline trees. With ensemble techniques, namely random forest and extra-trees, we achieved marginally better performance than with DT learning. However, it has to be considered that interpretability and explainability of the predictions decrease significantly with more complex methods compared to the original DTs: To explain why a certain prediction has been made, it is in practice possible to trace every step in a DT of depth 5. For an ensemble consisting of 100 full-grown DTs, this is very likely not the case. If it is however desired to further increase the prediction accuracy, the machine learning approach could be taken even further in future work: Instead of using hand-crafted features as input for classification, i.e. the limitation scores in our case, classifiers like neural networks could be trained directly on the temporal data extracted form the TCP header files.

References

[1] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and D. Collange, "A root cause analysis toolkit for tcp," *Computer Networks*, vol. 52, no. 9, pp. 1846–1858, 2008.

[2] L. J. Stemplinger, "Tcp flow performance root cause monitoring," Bachelor's Thesis, Technical University of Munich, 2019.

[3] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data mining and knowledge discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[4] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 309–322.

[5] M. Siekkinen, "Root cause analysis of tcp throughput: Methodology, techniques, and applications," in *PhD thesis, Institut Eurécom/Université de Nice-Sophia Antipolis, Sophia Antipolis*, 2006.

[6] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring tcp connection characteristics through passive measurements," in *IEEE INFOCOM 2004*, vol. 3. IEEE, 2004, pp. 1582–1592.

[7] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, "A machine learning approach to tcp state monitoring from passive measurements," in *2018 Wireless Days (WD)*. IEEE, 2018, pp. 164–171.

[8] ——, "Recurrent neural network-based prediction of tcp transmission states from passive measurements," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–10.

[9] I. El Khayat, P. Geurts, and G. Leduc, "Improving tcp in wireless networks with an adaptive machine-learnt classifier of packet loss causes," in *International Conference on Research in Networking*. Springer, 2005, pp. 549–560.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] K. P. Murphy, *Machine Learning. A Probabilistic Perspective*. The MIT Press, 2012.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[15] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[16] J. Bala, J. Huang, H. Vafaie, K. DeJong, and H. Wechsler, "Hybrid learning using genetic algorithms and decision trees for pattern classification," in *IJCAI (1)*, 1995, pp. 719–724.

[17] A. Papagelis and D. Kalles, "Ga tree: genetically evolved decision trees," in *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*. IEEE, 2000, pp. 203–206.

[18] S.-H. Cha and C. C. Tappert, "A genetic algorithm for constructing compact binary decision trees," *Journal of pattern recognition research*, vol. 4, no. 1, pp. 1–13, 2009.

doi: 10.2313/NET-2019-10-1_13

Virtio-Vsock - Configuration-Agnostic Guest/Host Communication

Johannes Wiesböck, Johannes Naab, Henning Stubbe
Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: johannes.wiesboeck@tum.de, {naab, stubbe}@net.in.tum.de

***Abstract*—Virtio-vsock provides zero-configuration communication channels to exchange data between a host and virtual machines running on the host. It builds upon the Socket API and the new addressing format AF_VSOCK, which allows easy porting of network applications to virtio-vsock. This paper explains the fundamentals of the new address format, and shows a flexible approach, which enables existing network applications to use virtio-vsock. This approach does not implement vsock support in every application but instead uses inetd-style socket activation to be applicable for many existing applications without modifying their source code. We focus on providing SSH connections to virtual machines over virtio-vsock, which will allow access to the virtual machines with almost no configuration. Additionally, we provide a generic solution for TCP-based applications.**

***Index Terms*—virtio, vsock, virtual machine socket, guest/host communication, ssh**

1. Introduction

Virtio-vsock is a zero-configuration communication interface, which enables data exchange between a host and virtual machines (VMs) running on it. It is designed to be available on a system by default without any configuration required. Further, it is based on the Socket API, which is also used for traditional network protocols.

Possible use-cases for this communication channel are guest agents, which run in a VM and interact with the host system, like the qemu-guest-agent [1]. Another use-case is to provide a host service to a VM like a remote file system. In the implementation part of this paper we will focus on running SSH connections over virtio-vsock. The goal of running SSH over a vsock connection is to provide an administration interface to VMs, which is independent of a network configuration. Thus, less configuration is needed and the interface can work more reliably.

This paper is structured as follows: First, we introduce the core concepts of virtio-vsock, which include both, high-level features, such as the address format, and implementation details like the underlying protocol. Next, we compare virtio-vsock with other alternatives for host/guest communication and we also show a present project that uses virtio-vsock. After that we describe an approach for using existing protocols such as SSH or HTTP over the vsock communication channel. Last, we evaluate the implementation and give a conclusion of our work with virtio-vsock.

2. Fundamentals of Virtio-vsock

This section introduces the basic concepts of virtio-vsock and implementation details.

2.1. Addressing Scheme

With VM sockets a new address format for the `socket()` system call named AF_VSOCK is added. An AF_VSOCK address is a 2-tuple consisting of a Context Identifier (CID) and a port. A unique CID is assigned to the host and to every VM in order to identify the individual machines. The CID is implemented as a 32-bit integer given in host byte order. Table 1 gives an overview of CIDs including CIDs which are reserved for special purposes [2].

The port of a vsock address is used to differentiate between multiple services running on one machine. Port numbers are implemented as 32-bit integers in host byte order [2], unlike TCP/UDP port numbers, which use 16-bit integers in network byte order. Port numbers below 1024 are called privileged. Only root can bind a socket to the privileged ports.

2.2. Socket Creation

Vsock connections can be managed using the Socket API. Thus, a VM socket can be created by a call to the `socket()` system call.

```
vsock = socket(AF_VSOCK, socket_type, 0);
```

According to version 5.2 of the Linux kernel source code [3], the only supported value for `socket_type` is SOCK_STREAM. This type provides reliable and stream-based communication with guaranteed and ordered delivery.

CID	Alias	Purpose
0	VMADDR_CID_HYPERVISOR	hypervisor
1	VMADDR_CID_RESERVED	reserved
2	VMADDR_CID_HOST	host
$[3; 2^{32} - 2]$	-	virtual machines
$2^{32} - 1$	VMADDR_CID_ANY	binding

TABLE 1: Overview of special CIDs

doi: 10.2313/NET-2019-10-1_14

Option	Description
REQUEST	initiate connection
RESPONSE	acknowledge connection initiation
RST	connection reset or address not bound
SHUTDOWN	request connection shutdown
RW	application data
CREDIT_UPDATE	updated credit data
CREDIT_REQUEST	explicitly request a credit update

TABLE 2: Overview of operations

2.3. Implementation Details

This section provides an overview of the protocol, which is used in vsock connections.

2.3.1. Flow Control. The stream-mode of virtio-vsock features a credit-based flow control mechanism, which prevents the sender from overloading the receiver [4]. The receiver informs the sender about its absolute amount of allocated receive buffer (buf_alloc) with every packet sent back or implicitly with a CREDIT_UPDATE packet, which is introduced in Table 2. The receiver also informs the sender about the amount of data, which was already forwarded to the application (fwd_cnt). Additionally, the sender keeps track of the absolute amount of data it has sent to the receiver (tx_cnt). Using this information, the sender can calculate its credit, which is the maximum amount of data it may send without overflowing the receivers buffer:

$$credit = buf\_alloc - (tx\_cnt - fwd\_cnt) \quad (1)$$

If the credit limit is reached, writing to the socket blocks until the receiver updates the fwd_cnt value.

2.3.2. Protocol. In this section we describe the lifetime of a stream-based virtio-vsock connection together with the operations involved in the connection. An overview of all possible operations is shown in Table 2. A connection consists of two endpoints, a server and a client, where the server runs on the VM and the client on the host or vice-versa.

We will now look into the steps involved in a possible vsock connection. Therefore, we will first look into the connection from an application point of view and later from the protocol point of view. First, the client application initiates the connection. Second, the client sends data to the server and third, closes the connection. These three phases are visualized as colored areas in Figure 1. Originally virtio-vsock used a different protocol than the one shown in this section. However, according to Hajnoczi [5] virtio-vsock protocol was partially reworked from the original protocol shown in [1]. The description of the protocol in this section was derived from observations made while examining connections using the packet analyzer Wireshark.

As shown in Figure 1, a connection is initiated with a two-way handshake. It begins with the client sending a packet of type REQUEST. If the server accepts the connection, it answers with an RESPONSE packet. The connection is now established. Application data is sent in a packet of type RW. Every time received data is forwarded to the application, for example when the server application reads data from the socket, the server sends a credit update to the client. The credit update informs the client about the updated *fwd_cnt* value. The connection is terminated with a two-way tear-down. The disconnecting side sends a packet of type SHUTDOWN, which is acknowledged with a RST packet terminating the connection.

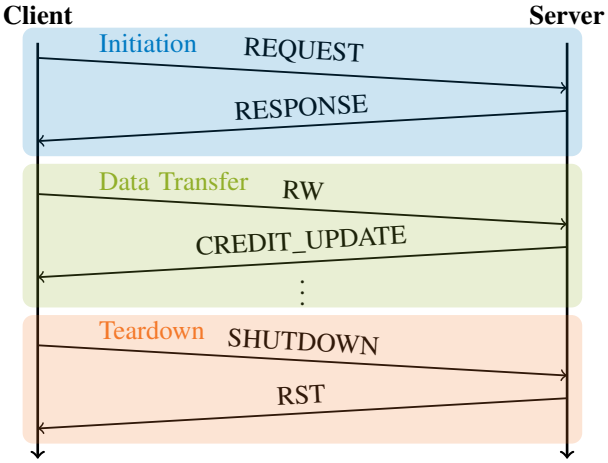


Figure 1: Overview of a sample stream-based vsock connection.

2.4. History

AF_VSOCK has originally been introduced to the Linux kernel in 2013 by VMware for VMware virtualization products [6]. AF_VSOCK was later implemented in virtio to be used with the kernel-based virtual machine (KVM) and QEMU. Virtio-vsock is part of the mainline Linux kernel since version 4.8 [7]. Support for virtio-vsock was added to libvirt in version 4.4.0 [8].

3. Related Work

This section will give a brief overview of alternatives to the virtio-vsock technology. Also an example where virtio-vsock is used in practice is covered in this section.

3.1. Alternatives to Virtio-vsock

Virtio-vsock can be compared to other technologies providing communication services between hosts and VMs. Two alternatives shown by Hajnoczi [1] are virtio-serial and virtual networking.

3.1.1. Virtio-serial. Virtio-serial is a virtual serial device, which is used to establish connections between hosts and guests [9]. A respective serial device is available on the guest and on the host-side. Applications can open the device and exchange data through the serial connection.

doi: 10.2313/NET-2019-10-1_14

Compared to virtio-vsock, virtio-serial has a few disadvantages [1]. The first downside is the limited number of channels, which equals the limited number of provided serial devices. To cope with this problem, data would have to be multiplexed on the application layer. Another disadvantage of virtio-serial is its implementation as a serial device. While this is not a problem per-se, it makes porting networking applications based on the Socket API more difficult than reusing the Socket API.

3.1.2. Networking. Another approach for guest/host communication is the usage of a virtual network [10]. This solution provides full network functionality to VMs. Thus, it is not only usable for guest/host communication but it also provides inter-VM networking and internet-access. The virtual network enables network applications to run between VMs without modifying them.This is possible, as the virtual network uses the internet protocol (IP) and thus supports all IP based applications. The downside of the networking approach is that creating network interfaces on the host and on the guests can be very complex and may not be desired [1]. In our case, we explicitly want to avoid additional network interfaces on the guest side, because they might influence the results of network-related tests or benchmarks running on the VMs.

3.2. NFS-vsock - File System over AF_VSOCK

Stefan Hajnoczi proposed support for the network file system (NFS) in 2016 [11]. The goal is to support NFS over vsock connections natively in the NFS implementation of the Linux kernel. For example, NFS over vsock could be used for network attached storage (NAS) services in cloud environments or to provide files to VMs during installation. Unfortunately, patches for vsock support in NFS have not been applied to the mainline Linux kernel so far, so using it requires a patched kernel.

4. Implementation

In this section we present the motivation for our implementation and possible implementation approaches. We select one approach and implement it for use with SSH and other protocols, such as HTTP and SMB.

4.1. Motivation

The motivation for this implementation is to enable various applications to use virtio-vsock for transport between hosts and VMs. We specifically focus on running SSH connections over VM sockets to provide a zero-configuration interface for VMs that is independent of a network configuration. Besides the SSH solution, a generic solution for TCP based services is also provided.

4.2. Approaches

In the following we compare two possibilities to enable applications to use virtio-vsock, namely native support and inetd-style.

4.2.1. Native Support. As stated in Section 2.2, AF_VSOCK reuses the existing Socket API, which simplifies the porting of network applications, as it should only require minor changes to the source code. By changing the first parameter of the call to `socket()` to AF_VSOCK and by updating the addresses accordingly, a network application could be ported. In many cases, this might not be sufficient to port the entire application, as only the networking part of an application can be ported easily, which might not apply to the entire application. An application, which is tightly bound to the TCP/IP protocol stack, can use the network configuration and the address format internally. Therefore, changes to the application logic are required to enable the AF_VSOCK format. Also user interfaces may be influenced when an additional protocol should be implemented. In general, porting an application natively to AF_VSOCK is not a trivial task and must be done for every application separately. Changes have to be made to the server and the client software respectively. A native implementation of AF_VSOCK support can be complex and therefore requires a lot of application knowledge.

4.2.2. Inetd-Style. An approach that can be applied to many services without modifying the application code is known as inetd-style socket activation.

When using socket activation, a super-server is set up to listen to incoming connections on a configured port. When a client connects to this port, the super server will accept the connection, start the actual application server and pass the connected socket to the application server. In inetd-style socket activation the connected socket is passed to the application by setting the servers standard input and output to the connected socket. In this scenario, the application server is not involved in the connection establishment and can be provided with a connected VM socket to communicate over AF_VSOCK. A possible super server is `systemd` which supports VM sockets since version 233 [12].

A disadvantage of the inetd-style is that it is not optimized for a specific application and thus has restrictions. Most importantly, it is required that an application server supports inetd-style socket activation. Another restriction is that additional ports can not be opened on behalf of the application, since all relays have to be set up in advance. This would make it unusable for example for FTP, which opens additional ports while in operation. Advantages of inetd-style are a simple implementation and support for many different services.

4.3. SSH

As mentioned in Section 4.1, we have a large interest in running SSH connections over VM sockets. SSH offers many features which can be used to enable this ability without port-forwarding and without modifying the source code of the SSH components. We show these features and how they are used in the following sections. Figure 2 shows the connection establishment, when a SSH client on the host tries to connect to a server running on a VM.

doi: 10.2313/NET-2019-10-1_14

```
ssh -o ProxyCommand='socat - SOCKET-CONNECT:40:0:x0000x16000000x04000000x00000000' user@vm
```

Listing 1: SSH command to connect to a server over a vsock connection

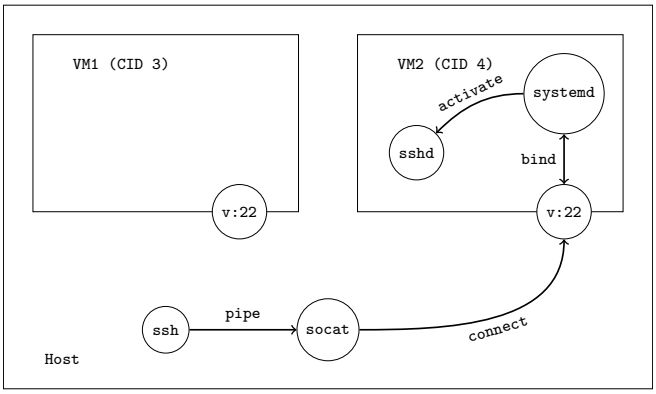


Figure 2: Establishment of a SSH connection from the host to a guest over AF_VSOCK

4.3.1. Client-Side. For the client, we use the widely installed OpenSSH client, which offers the `ProxyCommand` command line parameter. `ProxyCommand` can be any application that can connect to a remote SSH server and forward traffic coming from its standard input to the server and vice-versa. This proxy application will then connect to the destination server and forward all traffic from the client to the server and vice-versa. If this parameter is used, all SSH traffic is then passed through this proxy instead of the usual network connection. This creates the possibility to use `socat` [13] to forward the SSH connection to the destinations vsock. A possible command to connect to a SSH server using this method is shown in Listing 1. Here, `socat` connects to the SSH server running on port 22 of CID 4 and forwards this connection to the SSH client. Since `socat` does not offer special syntax for AF_VSOCK, the generic syntax has to be used. After `SOCKET-CONNECT`, `socat` is instructed to use protocol number 40 (AF_VSOCK) and type 0. After that, a hexadecimal representation of `struct sockaddr_vm` is given, which contains the port 22 (0x16) and the CID 4 (0x04) of the destination.

4.3.2. Server-Side. On the server-side, the SSH server `sshd` is started using inetd-style socket activation provided by `systemd` as explained in Section 4.2.2. This way, `systemd` is listening to incoming SSH connections on a local vsock port. Once a connection arrives on this port, systemd will accept the connection and start `sshd`. The connected file descriptor, which represents the accepted SSH connection, is passed to `sshd` as its standard input and standard output. `Sshd` is now able to use the SSH connection without being involved in the establishment of the connection.

4.4. Generic Solution using Port-Forwarding

Many existing network applications have no native support for the vsock protocol. Therefore, we implemented a generic solution that can be used by many applications using the TCP protocol but do not support vsock. It is not necessary for the application server to support inetd-style socket activation. This solution works by mapping vsock addresses to local IPv6 addresses. Thus, applications which are restricted to use TCP connections can access the vsock protocol over the interface introduced in Section 4.4.2.

4.4.1. Address Mapping. To make the vsock protocol available to applications, which support only TCP connections, CIDs of the vsock domain are mapped to local IPv6 addresses. For this mapping we use the IPv6 subnet fc00::/7, that is assigned for unique-local-unicast addresses, which are not routed on the internet. IPv6 addresses from this subnet can be chosen for local usage without colliding with globally unique addresses. Before CIDs can be mapped to IPv6 addresses, a random /64 prefix is chosen from the subnet fc00::/7. By definition, a locally assigned prefix from this subnet should have its eighth bit set to one [14]. Therefore, a possible valid prefix would be fd00:abcd:ef12:3456::/64. After the prefix is chosen, a CID is mapped to the IPv6 address space by adding the value of the CID to the prefix. In this example, this would result in CID 3 being mapped to the IPv6 address fd00:abcd:ef12:3456::3 and vice-versa.

Port numbers are mapped to TCP ports without changes if possible. Since AF_VSOCK offers 2^{32} different port numbers, all 2^{16} TCP ports can be directly mapped to vsock ports. Thus, if a service known from TCP is offered over vsock, its well-known port number can be reused. For example, a web server which usually listens to TCP connections on port 80 or 443, should also be available on the same vsock ports.

4.4.2. Forwarder Implementation. The implementation of the generic forwarder extends the concept in Section 4.2.2 by adding a relay to the setup that can forward traffic from TCP to vsock connections and vice-versa. Also in this implementation `socat` [13] is used as a relay software.

To add a relay to a TCP based server software, a socket activated instance of `socat` is configured on the

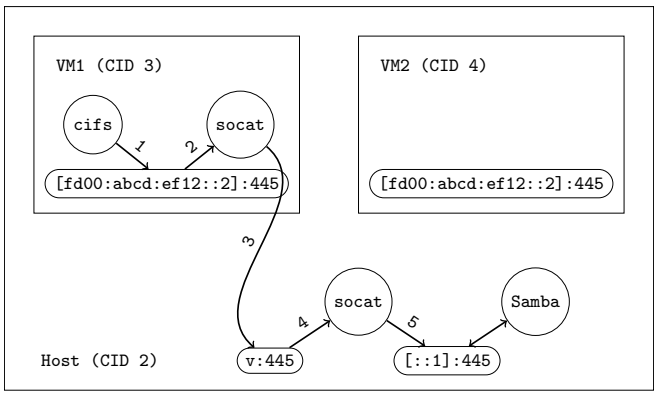


Figure 3: Establishment of a connection using two `socat` relays.

doi: 10.2313/NET-2019-10-1_14

servers machine. The configured VM socket is monitored by `systemd` for incoming connections. Once a client connects to the monitored port, `systemd` will start a `socat` instance and pass the connected vsock to it. The `socat` relay will then connect to the actual server listening on a local TCP port to establish the connection.

The client-side forwarder utilizes the address mapping introduced in Section 4.4.1. A client, which only supports TCP can connect to a `socat` relay via TCP. This relay can then forward the connection to the destination server via vsock. This socket activated relay is listening on a IPv6 address corresponding to the destination machines CID. If a client connects to a port on this special IPv6 address, the address will be translated into the corresponding CID. The `socat` relay will then connect to the given CID and to the port and forward traffic from the client to the server running on this address.

Figure 3 illustrates a possible scenario, where a Samba server is running on the host machine providing a file sharing service to VMs. If the SMB client software `cifs` tries to connect to the SMB server running on the host with CID 2, it actually connects to the IPv6 address fd00:abcd:ef12::2 representing this CID. Once a connection arrives on this socket, a `socat` instance is started via socket activation. This `socat` instance forwards the connection to the host over a vsock connection. The TCP port number is reused for VM sockets and is thus 445 for TCP as well as for vsock. When the host receives an incoming connection on vsock port 445, it will also start a `socat` relay to forward the traffic from this port to the SMB server listening on the local host on port 445. The connection is now established and data can be exchanged between client and server.

4.5. Evaluation

The proposed concept was successfully tested with SSH and worked reliably. The generic solution shown in Section 4.4 was tested with HTTP and SMB. HTTP was tested using a nginx web server running on a VM and a browser on the host. The SMB test setup included a Samba server running on the host system and a volume mounted in a VM over the vsock forwarder.

4.5.1. Performance. The achievable throughput of the forwarder was evaluated using the tool iperf3 with patched-in support for vsock connections [15]. For comparison, also the throughput of both, a virtual network interface and of a raw vsock connection was evaluated in addition to the forwarder. The base system for evaluation was a Lenovo ThinkPad T430 with a Intel Core i5-3320M CPU clocked at 2.60 GHz. The iperf3 server was running on a VM and the client was running on the host. In an iperf3 run with a duration of ten seconds, the virtual network connection achieved an average throughput of 14.2 Gbit/sec. The average throughput of a native vsock connection was 12.9 Gbit/sec on average and thus slightly slower than the network connection. In contrast to these comparably high values the forwarder setup using two `socat` relays only achieved an average throughput of 1.5 Gbit/sec. The considerably lower throughput may be caused by the multiple times that data has to be copied between buffers and the additional protocols involved.

4.5.2. Security Considerations. During development various connection scenarios were tested. As intended, we were not able to establish vsock connections between two VMs but only between the host and one VM. One exception is loopback connectivity. It is possible to connect from a VM to the same VM via AF_VSOCK. While this may be desired behaviour, it is to note that services, which are exposed over vsock, have to be secured properly if they should not be accessible from the VM itself. An example scenario would include a VM that should be configured via SSH over vsock. For configuration, a client must be able to connect as root over this SSH interface. Other than for configuration, the VM is operated by a untrusted user, who should not have root access to the machine. Because of the loopback connectivity, the user can connect to the local SSH server over vsock, which makes it necessary that the access is secured properly with a password or preferably with public keys. Without loopback connectivity, the SSH server would only be accessible from the host machine an thus could not be used by the user working on the VM. This might give the opportunity to omit authentication for SSH connection form the vsock interface, since it could only be access from the host machine. Loopback connectivity was explicitly removed by Google for ChromeOS [16] to prevent applications from connecting to other applications on the same machine. Loopback connectivity is present in the mainline Linux kernel and removing it would require a patched kernel.

5. Conclusion

This paper gave a short introduction to the virtio-vsock technology. We showed, that virtio-vsock provides a reliable and user-friendly communication mechanism for VM setups. We were able to enable vsock support for different services using inetd-style socket activation.

Even though socket activation worked for all tested services, we would like to see native support for AF_VSOCK connections in the future, as it might outperform the shown implementation using two `socat` relays. Native support would also make it easier to use network applications between hosts and VMs, with zero-configuration. So far, desirable features such as support for NFS are not part of the mainline Linux kernel, which would require building a custom kernel. Together with the security considerations in Section 4.5.2, it has to be considered if this effort is worth the benefits gained in features and security. Future research should investigate, if it is possible to increase the throughput of the forwarder-setup, possibly by investigating if a specifically developed and tuned forwarder would perform better than `socat`.

References

[1] S. Hajnoczi, “virtio-vsock Zero-configuration host/guest communication,” Accessed on: 2019-06-05. [Online]. Available: https://vmsplice.net/~stefan/stefanha-kvm-forum-2015.pdf

doi: 10.2313/NET-2019-10-1_14

[2] *man 7 vsock*, Accessed on: 2019-08-26. [Online]. Available: http://man7.org/linux/man-pages/man7/vsock.7.html

[3] "virtio_vsock.h (kernel version 5.2)," Accessed on: 2019-06-01. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/uapi/linux/virtio_vsock.h?h=v5.2

[4] A. He, "Introduce VM Sockets virtio transport," *LWN.net*, 2013, Accessed on: 2019-06-01. [Online]. Available: https://lwn.net/Articles/556550/

[5] S. Hajnoczi, "Add virtio transport for AF_VSOCK," *LWN.net*, 2016, Accessed on: 2019-06-01. [Online]. Available: https://lwn.net/Articles/695981/

[6] A. King, "VSOCK: Introduce VM Sockets," Accessed on: 2019-09-01. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d021c344051af91f42c5ba9fdedc176740cbd238

[7] S. Hajnoczi, *Features/VirtioVsock*, Accessed on: 2019-06-09. [Online]. Available: https://wiki.qemu.org/Features/VirtioVsock

[8] "libvirt: Releases," Accessed on: 2019-06-09. [Online]. Available: https://www.libvirt.org/news.html

[9] A. Shah, *Features/VirtioSerial*, Accessed on: 2019-06-12. [Online]. Available: https://fedoraproject.org/wiki/Features/VirtioSerial

[10] *Documentation/Networking*, Accessed on: 2019-06-12. [Online]. Available: https://wiki.qemu.org/Documentation/Networking

[11] S. Hajnoczi, "NFS over virtio-vsock Host/guest file sharing for virtual machines," Accessed on: 2019-06-22. [Online]. Available: https://vmsplice.net/~stefan/stefanha-connectathon-2016.pdf

[12] "Systemd NEWS," Accessed on: 2019-06-13. [Online]. Available: https://github.com/systemd/systemd/blob/v233/NEWS#L303

[13] *man 1 socat*, Accessed on: 2019-08-26. [Online]. Available: https://linux.die.net/man/1/socat

[14] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," Internet Requests for Comments, RFC Editor, RFC 4193, October 2005.

[15] S. Garzarella, "iperf," Accessed on: 2019-08-27. [Online]. Available: https://github.com/stefano-garzarella/iperf-vsock

[16] "Chrome OS source: virtio_transport.c," Accessed on: 2019-06-22. [Online]. Available: https://chromium.googlesource.com/chromiumos/third_party/kernel/+/refs/heads/chromeos-4.4/net/vmw_vsock/virtio_transport.c#188

doi: 10.2313/NET-2019-10-1_14

Deterministic Networking (DetNet) vs Time Sensitive Networking (TSN)

Xiaotian Yang, Dominik Scholz*, Max Helm *
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge25gad@mytum.de, scholzd@net.in.tum.de, helm@net.in.tum.de

***Abstract*—With increasing demands of highly reliable networks with bounded latency and low jitter, a lot of Ultra-Low Latency Network studies are in progress. This paper focuses on IEEE Time Sensitive Networking (TSN) and IETF Deterministic Networking (DetNet). We analyze the similarities and differences between these two networking standards and give a survey of the published standards and possible future work of the DetNet and TSN Task Groups.**

***Index Terms*—Time Sensitive Networking, Deterministic Networking, Ultra-Low Latency, Network Standards**

1. Introduction

Ethernet is a series of connectivity services, first standardized by IEEE 802.3 Ethernet Working Group (WG) in 1983. Ethernet has been widely adopted for regular Internet/Data center since the 1980s and for telecommunication and industry networks recently due to its cost-effectiveness and flexibility [1]. However, with the increasing demand on real-time capabilities in industrial applications, tranditional Ethernet lacks Quality of Service (QoS) metrics of end-to-end latencies. Therefore, markets and customers turn their attention to Ultra-Low Latency (ULL) networking standards, which can signaficantly reduce the latencies to milliseconds. [2]

Among these ULL networks, the IEEE Time-Sensitive Network (TSN) Task Group (TG) and the IETF Deterministic Network (DetNet) TG are of interest. These two TGs devote to providing deterministic networking standards with low bounded latency and high reliability. While TSN standards focus only on OSI model Layer 2 (LANs), the DetNet standards extend the technologies to Layer 3 (IP) [3]. This paper aims to give an overview over these two standards and discuss their current state and future.

The content of this paper will be organized as follows. In Section 2 we will discuss the previously published related work on this topic. The background studies i.e. the history of TSN TG and DetNet TG will be reviewed in Section 3. In Section 4 and Section 5, a basic overview of TSN and DetNet will be introduced. To get further insights into the research value, Section 6 demonstrates a range of practical use cases. As a next step, we will discuss the current state and the future work of TSN and DetNet in Section 7. Finally, the conclusion of this TSN and DetNet survey will be given in Section 8.

2. Related work

The documents on the official website of the TSN TG [4], [5] only cover the published standards or in progress projects of IEEE 802. The files of DetNet TG are currently just Internet drafts [6], [7], [8]. Here [6] is the IETF draft of DetNet problem statement. By contrast, a general Introduction to TSN has been published in the IEEE Communications Standards Magazine in [3]. Additionally, a survey on Ultra-Low Latency networking including TSN, DetNet and related 5G ULL Research has been presented in [9]. In order to differ from all these previous literatures, this paper provides a comparison of TSN and DetNet and tries to summarize their corresponding features and up-to-date progress.

Different from optimized Ehternet fieldbus, such as EtherCAT (Ethernet for Control Automation Technology), TSN or DetNet are add-ons to the best effort switched environment. Besides IEEE TSN and IETF DetNet, there are also some other related standards and researches in the field of ULL Network, e.g. Wireless High Performance (WirelessHp) and Multefire. [10] To meet the ULL requirements in industrial sites, specifications like WirelessHP are applied. The goal of Wireless HP is to realize multi-Gbps data rate aggregation and lower the packet transmission time within microseconds through physical layer solution. Moreover, Multefire, a Long-Term Evolution (LTE) based technology can also be used to boost data rate and reduce transmission time and latency. [10]

In contrast to these related studies, TSN and DetNet mainly focus on deterministic latency and specify a series of standards to achieve ULL and reliable networking over best-effort Ethernet networks.

3. Background Studies

The predecessor of TSN standards was Audio Video Bridging (AVB) industrial standards. Audio Video Bridging TG was established in 2007 by the IEEE 802.1 standards committee. This TG is chartered to specify time-synchronized low latency streaming services over IEEE 802 networks. AVB standards specify the implementation of a plug-and-play home or audio or video production studio but only operate in OSI model Layer 2. [3]

Motivated by the great success of AVB standards, the IEEE committee plans to expand AVB applications into industrial fields. Therefore, in 2012 the AVB TG was renamed as TSN TG [3]. The TSN TG aims to establish deterministic network services and to reduce the latency to microseconds or milliseconds to meet precise industrial control or automation demands [9]. TSN also only works in bridged Layer 2 networks.

With the increasing progress in TSN standards, people want such ULL Networking not just confined to Layer 2.

doi: 10.2313/NET-2019-10-1_15

In 2015 the IETF created its DetNet TG. The goal of the DetNet TG is to extend bounded latency, low latency jitter and highly-reliable services to both Layer 2 and Layer 3. [3]

As mentioned in Section 2, the TSN TG has already published a series of networking standards, while the documents of the DetNet TG are currently just Internet drafts and still in progress.

4. Overview of Time Sensitive Networking

This section provides a survey of the features and standards of TSN.

4.1. TSN Features

TSN TG is based on AVB TG and chartered to implement bounded latency, low latency jitter over traditional Ethernet. The significant characteristics of TSN Networks are as follows:

Time synchronization: In order to meet the requirements of real-time control or automation, time sensitive network is designed as a time-aware network. The clock of all devices in a Time-Sensitive network must be synchronized. This technique can be realized through various variants of existing timing specific protocols like IEEE 1588. One IEEE official offered synchronization standard is IEEE 802.1AS. Through network-wide shared time reference, TSN is capable to fix the transition delays and send packets at the time arranged. [9]

Bounded latency and zero congestion loss: Congestion happens when there are overflowing streams in a node that beyond the capability of the network. Network congestion is the main reason of packet loss and latency. [3] Thanks to buffer allocation, queuing algorithm and frame preemption, TSN achieves bounded low latency and zero congestion loss.

The principle of realizing zero congestion loss is computation of buffers in the worst-case. Figure 1 defines packet latency into five components: Output delay, Link delay, Preemption delay, Processing delay and Queuing delay. [3]

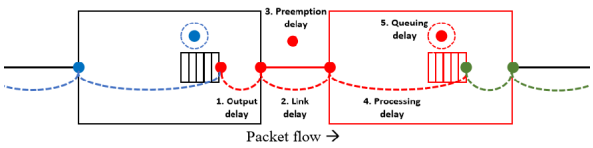


Figure 1: TSN Timing Model [3]

The uncertainties in networks during transmission like abrupt interruptions of packets and states of forwarding nodes lead to variability of these delay times. Buffers in the queue are allocated to compensate the variations of delay.

Additionally, the queuing delay is assumed to be calculable according to the queuing algorithm applied to TSN. Since the packet selection schedule of queuing algorithm applied to TSN are mathematically analyzable, the buffer requirement in the worst-case can be predicted. [3]

The queuing mechanism is mainly standardized in IEEE 802.1Qav, which aims to constrain the number of buffers required in a network. Credit-based shaper (CBS) is the key concept of the queuing algorithm. When there exists no frame in the queue, the credit is set to zero. The credits increase when a frame is added into the queue and decrease when a frame is sent. This mechanism allows a queue to transmit only if the credits are nonnegative and the channel is not occupied [9]. CBS also defines constraint parameters such as maximum frame size, maximum reference size and maximum port transition rate. Thus, the latency per bridge can be limited [3]. The flow chart in Figure 2 clearly illustrates the CBS operation for a given queue.

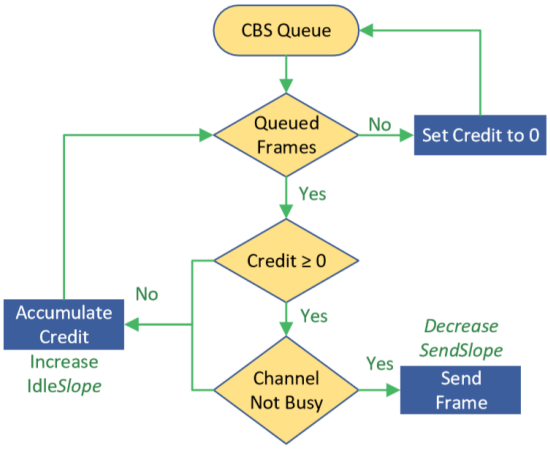


Figure 2: Flowchart of CBS operation for a given queue [9]

Another mentionable technique here is frame preemption, specified in IEEE Std 802.1 Qbu and IEEE 802.3 br. In IEEE 802.1Qbv a guard band is added in front of scheduled time-critical traffic to prevent low priority traffic from transmission when that transmission cannot be finished before the scheduled traffic window. The preemption mechanism enables midway stop of frame transmission before the start of a guard band and execute the transmission of another frame with higher priority. After accomplishment, the original transmission can continue. [9]

Ultra reliability: The core technique improving the reliability of TSN is frame replication and elimination. This technique reduces the packet loss caused by equipment failure in network. The procedures of frame replication and elimination are documented in IEEE 802.1CB: 1) number the sequence of packets and replicate them in the network, 2) identify the redundant and eliminate packages at or near the destination. [11]

Moreover, as shown in the second case of Figure 3, packets can also be re-replicated or eliminated at various nodes, like node B and node E. Thus, a failure of node A and E or node B and C will not affect the packet end-to-end delivery in TSN. Through this mechanism, TSN is capable to handle multiple errors and increase the transmission reliability.

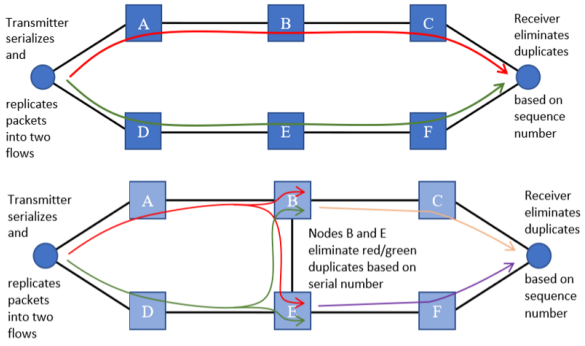


Figure 3: Mechanisms of packet replication and elimination [3]

4.2. TSN standards

Above survey only covers some key features of TSN. The TSN TG also standardizes many other meaningful techniques and mechanisms, these standards will be summarized in this subsection.

Most of these standards are amendments for IEEE 802.1Q-2018: Bridges and Bridged Networks. Project ID with capital letters indicates stand-alone documents, such as IEEE 802.1CM. Lower case letters in project ID means the standards are amendments, e.g. IEEE Std 802.1Qbu. [3] Based on introduced features in Section 4.1, this paper splits published standards into three groups as shown in Table 1:

TABLE 1: TSN published Standards Summary

Group	Standard Name	Features
Time Synchronization	IEEE 802.1AS	Timing models and clock synchronization for TSN
Zero Congestion Loss	IEEE 802.1Qbu	Frame Preemption
	IEEE 802.1Qbv	Enhancements for scheduled traffic
	IEEE 802.1Qch	Cyclic Queuing and Forwarding, amendment for IEEE Std 802.1Q
	IEEE 802.1Qcp	YANG Data Model
	IEEE 802.1CM	Time-Sensitive Networking for Fronthaul
	IEEE 802.1Qcc	Stream Reservation Protocol (SRP), amendment for IEEE Std 802.1Q
Reliability	IEEE 802.1CB	Frame Replication and Elimination for Reliability
	IEEE 802.1Qci	PerStream Filtering and Policing
	IEEE 802.1Qca	Path Control and Reservation

IEEE 802.1AS is based on IEEE 1588v2 and specifies the precise timing model and clock synchronization in TSN. Analog to Precision Time Protocol (PTP) defined by IEEE 1588, IEEE 802.1AS defines generalized PTP (gPTP), which enables synchronous transportation over all media. The prerequisite of IEEE 802.1AS is that all bridges and end stations in network should be time aware. Then, IEEE 802.1AS selects one system as grandmaster and assign the ports in the network as master, slave or passive roles. This helps to form a synchronization hierarchy in TSN. The grandmaster is expected to transmit synchronization information on slave ports and loops can be broken through passive role ports. This mechanism ensures the network wide clock synchronization. [12]

IEEE 802.1Qbv is standardized as enhancements to traffic scheduling Time-Aware Shaper (TAS). This standard helps to construct the well-defined QoS for TSN through specified TAS. The mechanism of this standard is as follows: 1) time-aware traffic windows are used to schedule the critical traffic streams, 2) a guard band is added before scheduled traffic windows to prevent transmission interrupted by lower priority frames. [9]

IEEE 802.1Qch Cyclic Queuing and Forwarding (CQF) aims to synchronize enqueue and dequeue operations in TSN. Through CQF synchronization, frames can be transmitted in a cyclic manner. And the network transit latency can be characterized by the cycle time. [9]

IEEE 802.1Qcp standardizes the YANG data model and utilizes Unified Modeling Language (UML) representation. YANG is a formalized data modeling language widely adopted in industries. Motivated by this, TSN TG decided to establish standards supporting YANG data modeling. IEEE 802.1Qcp is also applied to support other specifications. [9]

IEEE 802.1CM refers to TSN profiles for fronthaul, its application will be explained in Section 6.1.

IEEE 802.1CB specifies frame replication and elimination. The mechanism applied in this standard has been covered in Section 4.1.

5. Overview of Deterministic Networking

This section intends to offer an overview of the features and currently established internet drafts and RFCs of DetNet.

5.1. DetNet Features

The IETF DetNet TG has similar charters to the TSN TG. Therefore, DetNet also has features such as time synchronization, zero congestion loss and reliability like TSN. Additionally, DetNet devotes to extend the ULL and highly reliable services to layer 3 networks. DetNet TG also works on coexistence of DetNet with normal traffic and DetNet misbehavior mitigation. [7]

Time synchronization: Like in TSN, devices in DetNet should share common timing reference. DetNet time synchronization is realized through existing IEEE 1588 and IEEE 802.1AS.

Zero congestion loss and Reliability: Similar to TSN techniques stated in Section 4, ULL characteristics and zero congestion loss in DetNet are achieved through queuing algorithms, buffer reservation and packet preemption. Since queuing algorithms also fit well to routers, the number of buffers in the worst-case is analogously mathematically analyzable in DetNet. [9]

One difference in DetNet is that in order to get lower jitter, end-to-end latency DetNet has not only upper

doi: 10.2313/NET-2019-10-1_15

bounds but also lower bounds. The concrete methods of jitter minimization include: 1) network-wide time synchronization to sub-microsecond accuracy 2) count time-of-execution fields into the application packet. To ensure the reliability of DetNet, filters and policers are applied to detect failures and error of packets. When fault is detected, filters and policers will disrupt and adjust the transmission. Moreover, packet replication and elimination techniques are also applied in DetNet. [7]

To fix the coexistence issue with normal traffic, DetNet assigns critical flows with higher priority than best-effort flows. This will not threaten the network operation, since both critical flows and best-effort traffic have bounded latency and bandwidth in DetNet.

Security: Security considerations are another essential feature in DetNet. To achieve request security and control security of DetNet resources, authentication and authorization should be used for devices connected to a DetNet domain to ensure that the administrative configuration of parameters is constrained to authorized devices. [7]

Control of DetNet can be classified as centralized or distributed. For centralized control of DetNet, Abstraction and Control of Traffic Engineered Networks (ACTN) is used for security considerations. For distributed control of DetNet, security considerations are expected to be achieved through the security properties of the deployed DetNet protocols. [7]

5.2. DetNet Internet-drafts

Since Deterministic networking TG has no published standards, Table 2 lists up-to-date DetNet Internet drafts and RFCs. (June,2019) Some critical Internet drafts among them are selected to be explained further in this section.

***-architecture** introduces the overall DetNet architecture and the mechanisms used to achieve DetNet QoS. The DetNet QoS includes resource allocation, service protection and explicit routes. Similar to TSN, provision of sufficient buffer at each node and packet replication and elimination are also used in DetNet to ensure ULL services. [7]

***-data-plane-framework** introduces the framework for DetNet controller plane and its requirements. DetNet services are currently specified on IP networks or MPLS (Multiprotocal Label Switching) networks. Encapsulation in DetNet enables the flows to be transmitted to other data plane technology beyond its original stream type.

***-security** discusses security problems in DetNet and collects related considerations from other DetNet drafts. Security is highly important in DetNet, since DetNet which operates in higher OSI model layer owns more potential of cyber-attack. Various threats such as delay attack, path manipulation and their corresponding mitigations through path redundancy, encryption, performance analytics or DetNet node authentication are all analyzed in this documentation. [13]

DetNet TG has updated two RFCs in May 2019. RFC8557 (was draft-ietf-detnet-problem-statement) illustrates the necessity of establishing DetNet for industrial applications and RFC8578 (was draft-ietf-detnet-use-cases) describes a series of DetNet use cases in various fields.

TABLE 2: DetNet official Internet drafts Summary

draft-ietf-detnet	Features
-architecture	Introduce DetNet architecture and the used techniques to carry real-time unicast/multicast data streams
-data-plane-framework	Specify the framework for DetNet controller plane and its requirements
-dp-sol-ip	DetNet IP Data Plane Encapsulation
-dp-sol-mpls	DetNet MPLS (Multiprotocal Label Swichting) Data Plane Encapsulation
-flow-information-model	an overview of DetNet model for integration over Layer 2 and Layer 3
-ip	Describe how can DetNet operate over IP packet switched network
-ip-over-mpls	Describe how can DetNet operate in an IP over MPLS packet switched network
-ip-over-tsn	Describe how can DetNet operate in an IP over TSN
-mpls	DetNet Data Plane: MPLS
-mpls-over-tsn	DetNet Data Plane: MPLS over IEEE 802.1 Time Sensitive Networking
-mpls-over-udp-ip	DetNet Data Plane: MPLS over IP
-tsn-vpn-over-mpls	DetNet Data Plane: IEEE 802.1 Time Sensitive Networking over MPLS
-security	Deterministic Networking (DetNet) Security Considerations
-topology-yang	Deterministic Networking (DetNet) Topology YANG Model
-yang	Deterministic Networking (DetNet) Configuration YANG Model

6. Use Cases

Various applications in industries require deterministic flows, which is exactly the core of TSN and DetNet [8]. Besides, DetNet enables interconnection between Layer 2 and Layer 3. Therefore, TSN and DetNet use cases cover a wide range of industries including professional audio and video, control and automation systems, industrial machine-to-machine, vehicle applications etc. [8] We only introduce selected examples in this section, more details can be obtained from [3] and [8].

6.1. Use Case of TSN

One use case is the TSN applicability in 5G (5th generation mobile/wireless networks). TSN with bounded latency and high-reliability are necessary in 5G scenario.

TSN helps network slicing and realize the fronthaul connection in Ethernet bridged networks [9]. Network slicing implies that there is no interference between applications or users. Moreover, TSN techniques like resource reservation and traffic scheduling are helpful to aggregate

dataflow in 5G Bearer Networks. As shown in Figure 4 IEEE Std 802.1CM specifies TSN profiles for 5G fronthaul. [14] The detailed demonstration of 5G mechanism exceeds the scope of this survey.

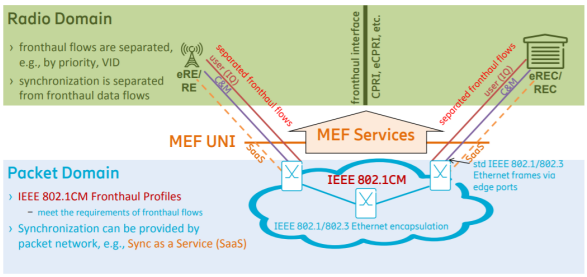


Figure 4: Fronthaul details with IEEE Std 802.1CM [14]

6.2. Use Case of DetNet

As DetNet standardization is still in progress, it has appealing potentials in industrial applications but no concrete examples currently. Therefore, this subsection only selects two typical applications from [8] to provide readers a general overview of DetNet use cases.

One promising DetNet use case is industrial Machine to Machine (M2M). Industrial M2M communication is mainly executed through Programmable Logic Controllers. DetNet in this use case is deployed to ensure the critical control/data flow is successfully delivered end to end within demanded time constraints. Industrial M2M with requirements like time synchronization, low packet loss, ultra-low delivery time and reliability are exactly corresponding to the characteristics of DetNet. [8]

Another typical use case is the professional audio and video industry (ProAV), including broadcast and music or film production. These industries are now faced with the transition to packet-based infrastructure and integration with IT infrastructure. With the support of DetNet services, ProAV applications will be able to interconnect Layer 2 and Layer 3 and then achieve broadcast over wider areas. [8]

7. Future work of TSN and DetNet researches

In contrast to DetNet researches, the TSN studies are currently isolated from external networks and only restricted to small-scale domains like in-vehicle networks. However, the industrial use cases of TSN such as Machine-to-Machine Communication or Industrial Control and Automation systems often are equipped with highly complex infrastructure. Further TSN studies may need to find a solution to enhance the interconnectivity and simplify the network management mechanisms. [9] Additionally, the standards considering security and privacy may also be interesting future research topics for TSN.

Since the IETF DetNet is a rising study field in recent years, DetNet architecture and standards still need a long way to be implemented and improved. For example, DetNet interconnection between Layer 2 and Layer 3 is realized with support of TSN LAN services, hence DetNet requires stable resource sharing techniques over Layer 2. Another important future study will be the concrete use cases of the integration of DetNet with traditional external networks.

8. Comparison of Time-Sensitive Networking and Deterministic Networking

IEEE TSN and IETF DetNet can be compared in the following aspects:

OSI Layer: The most important difference between TSN and DetNet is the OSI layer they operate on. While TSN is confined to Layer 2, DetNet extends the corresponding properties to Layer 3 or even higher layers.

Bounded Latency: Another difference is that in TSN only upper bound is predefined to reduce latency. However, in DetNet exist both upper and lower bounds to realize jitter minimization.

Data plane: DetNet nodes can connect to other subnetworks including MPLS Traffic Engineering (TE), IEEE TSN and Optical Transport Network (OTN). Also multilayer DetNet systems can be constructed in the future, which cannot be achieved with TSN. [14]

Security: DetNet TG also pays more attention on security considerations than TSN TG, since DetNet, which expands its scope to higher OSI model layer, is faced with higher cyber-attack probabilities.

Current status: The IEEE TSN TG has already specified and published a series of standards which have been adopted to concrete use cases. By contrast, the IETF DetNet TG is still immature and remains in its starting stage.

9. Conclusion

Both TSN and DetNet TG aim to create highly reliable ULL networks with features like time synchronization, zero congestion loss, reliability and security for real-time industrial applications. DetNet TG is chartered to expand TSN mechanisms beyond Layer 2 LANs to higher layer, for example, time synchronization techniques and frame replication and elimination mechanisms are deployed both in TSN and DetNet. This survey also discusses the possible future studies of these two networks include the enhancement of interconnectivity and further security considerations. With increasing researches in TSN and DetNet, more and more concrete use cases in the field of industrial M2M communication and ProAV etc. will be implemented in the future. Despite some limitations of current standards or internet drafts, TSN and DetNet combined with optimized Ethernet fieldbus such as EtherCAT will impact the traditional IEEE 802 networks significantly.

References

[1] R. Santitoro, " Metro Ethernet Services – A Technical Overview," 2003.

[2] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and Industry 4.0," vol. 11, no. 1, pp. 17–27, 2017.

[3] N. Finn, “Introduction to Time-Sensitive Networking,” vol. 2, no. 2, pp. 22–28, 2018.

[4] IEEE LAN/MAN Standards Committe, “IEEE Standard for Local and metropolitan area networks—Audio Video Bridging (AVB) Systems ,” pp. 1–45, 2011.

[5] ——, “IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Network,” pp. 1–1832, 2014.

[6] N. Finn and P. Thubert, “Deterministic Networking Problem Statement,” 2019, unpublished.

[7] ——, “Deterministic Networking Architecture,” 2019, unpublished.

[8] E. E. Grossman, “ Deterministic Networking Use Cases,” 2018, unpublished.

[9] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, “Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research,” pp. 1–59, 2018.

[10] X. Jiang, H. S. Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, “Low-Latency Networking: Where Latency Lurks and How to Tame It,” pp. 1–24, 2018.

[11] IEEE LAN/MAN Standards Committe, “IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability,” pp. 1–102, 2017.

[12] Geoffrey M. Garner and Hyunsurk Ryu, “Synchronization of Audio/Video Bridging Networks Using IEEE 802.1AS,” pp. 1–8, 2011.

[13] T. Mizrahi, E. E. Grossman, A. Hacker, S. Das, J. Dowdell, H. Austad, K. Stanton, and N. Finn, “Deterministic Networking Security Considerations,” 2019, unpublished.

[14] "Time Sensitive Networking for 5G". [Online]. Available: https://datatracker.ietf.org/meeting/103/materials/slides-103-dmm-time-sensitive-networking-for-5g-01

doi: 10.2313/NET-2019-10-1_15

ISBN 978-3-937201-68-9
DOI 10.2313/NET-2019-10-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)