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

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

Winter Semester 2018/2019

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)

Winter Semester 2018/2019

Munich, July 20, 2018 – February 17, 2019

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET 2019-06-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Winter Semester 2018/2019

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 WS 18/19
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, July 20, 2018 – February 17, 2019
ISBN: 978-3-937201-64-1

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2019-06-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2019-06-1
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 Winter Semester 2018/2019. 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 Daniel Meint with the paper *From FIFO to Predictive Cache Replacement* and Jonas Andre with the paper *Open vSwitch Configuration for Separation of KVM / libvirt VMs*.

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

We hope that you appreciate the contributions of these seminars. If you are interested in further information about our work, please visit our homepage `https://net.in.tum.de`.

Munich, May 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

Fabien Geyer (fgeyer@net.in.tum.de)
Technical University of Munich

Stephan Günther (guenther@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

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

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

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

Cora Perner (clperner@net.in.tum.de)
Technical University of Munich

Minoo Rouhi (rouhi@net.in.tum.de)
Technical University of Munich

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

Marcel von Maltitz (maltitz@net.in.tum.de)
Technical University of Munich

Seminar Homepage

`https://net.in.tum.de/teaching/ws1819/seminars/`

Contents

Block Seminar

Case Study and Practical Assessment of BPMN with Camunda 1
Vincent Bode (Advisor: Marcel von Maltitz, Holger Kinkelin)
Performance of Secure Multiparty Computation 5
Ludwig Dickmanns (Advisor: Marcel von Maltitz)
Overview of TCP Congestion Control Algorithms 11
Moritz Michael Geist (Advisor: Benedikt Jaeger)
Robustness of Scanner Exams with TUMexam 17
Simon Kassahun (Advisor: Stephan Günther)
Network Resource Management for Virtual Networks with Learning Algorithms 21
Anton Mai (Advisor: Cora Perner)
From FIFO to Predictive Cache Replacement 25
Daniel Meint (Advisor: Stefan Liebald)
Time Sensitive Networking for Wireless Networks – A State of the Art Analysis 33
Alexander Mildner (Advisor: Fabien Geyer)
Measuring TCP Performance Metrics with Bro 39
Leonhard Josef Stemplinger (Advisor: Simon Bauer)

Seminar

Open vSwitch Configuration for Separation of KVM/libvirt VMs 43
Jonas Andre (Advisor: Johannes Naab)
Networking in MirageOS 47
Fabian Bonk (Advisor: Paul Emmerich)
Bot-based IT Troubleshooting 53
Benjamin Braun (Advisor: Jonas Jelten, Simon Bauer)
Client Monitoring with HTTPS 59
Felix Hartmond (Advisor: Simon Bauer)
Caching with Relation 63
Mohamad Nour Moazzen (Advisor: Stefan Liebald)
Investigating TCP SYN Flood Mitigation Techniques in the Wild 67
Julian Villing (Advisor: Minoo Rouhi, Dominik Scholz)
Networking in Biscuit 71
Sebastian Voit (Advisor: Paul Emmerich)
Recent Progress on the QUIC Protocol 77
Mehdi Yosofie (Advisor: Benedikt Jaeger)

Case Study and Practical Assessment of BPMN with Camunda

Vincent Bode, Marcel von Maltitz*, Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: vincent.bode@tum.de, vonmaltitz@net.in.tum.de, kinkelin@net.in.tum.de

***Abstract*—Processes are a sequence of activities designed to reach a targeted outcome. In this general form, processes are an essential component of many types of systems. Process modeling is a technique that formalizes processes by documenting them using abstract notation. It can be used to improve a running system by optimizing the modeled process. Using a case study process of scheduling an appointment with a professor, we elicit some requirements and discuss the benefits and drawbacks of using Business Process Model and Notation (BPMN), a widespread standardized notation for modeling processes [5], for implementing this process. We observe that the formal BPMN process is a viable alternative to less formalized solutions, that it has potential for automation and a reduction of errors, but it can incur higher maintenance effort than an informal version. The applicability of BPMN therefore varies across use cases.**

***Index Terms*—process modeling, bpmn, camunda**

1. Introduction

Processes are "a series of actions or operations conducing to an end" [1] found in many systems. Whether they are defined explicitly or implicitly, achieving a target goal is always driven by a process. They can be categorized according to specific traits such as their run-time, execution frequency or whether the process is mainly human or machine based. Ensuring that frequent and time-intensive processes are implemented efficiently and effectively can have a positive impact on the system's performance. This is the field of process modeling, where the interactions between participants, tasks and communication events are analyzed and optimized with the goal of improving certain aspects of the system, such as processing speed, fault-tolerance or reliability.

The rest of the paper is divided into the following sections. Section 2 motivates process modeling. The case study is introduced in Section 3 and some requirements are stated. Section 4 introduces BPMN and Camunda used for reimplementing the process in a formalized manner. A review of the benefits and challenges encountered during the implementation is presented in Section 5. Finally, the solution is evaluated in Section 6.

2. Reasons for Process Modeling

Implicit processes are loosely defined and can consist of nothing more than a state and a target outcome, while explicit processes are fixed in documentation or formalized interaction. Making processes explicit has several advantages. For one, it allows an analyst to gain insight to complex workflows, allowing them to analyze "the sequence of steps involved in moving from the beginning to the end of a working process" [1] and making the process observable. This allows the analyst to perform studies that measure the workflow's performance and compare it to different variations, allowing them to improve the process incrementally. A second advantage is that the process model is also a way of documenting how a process works, allowing for standardization, repeatability, simpler knowledge transfer and increased transparency. One disadvantage is that a process model is also an artifact that needs to be kept up to date. To learn about the impact of formalizing processes, we wish to examine a specific problem involving human and machine participants.

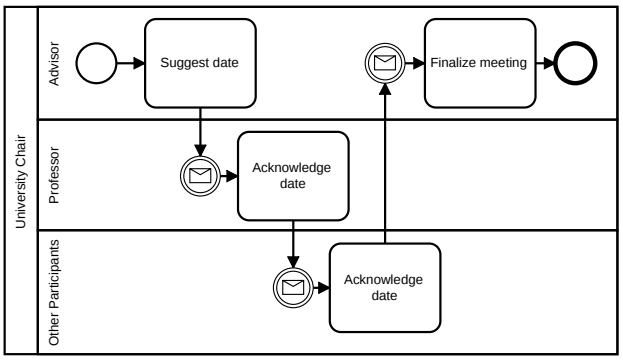

Figure 1. A schematic of the current scheduling of a meeting process. It shows the best-case scenario, where each participant is available at the suggested date and all communication is error free.

3. Case study

Throughout this work, a case study will be used to evaluate the practical effects of using process modeling on a process to improve its performance. We will examine the scheduling of an appointment with a professor and multiple other participants at a university chair. A schematic of this process may be seen in Figure 1. Table 1 shows multiple issues that affect the performance of the current process. To support our modeling and implementation, a workflow management system will be used. Such a system can administer formalized processes and track the state and history of any process running inside of it. The case study focuses on how the current problems can be solved by introducing such a system and which new ones may arise.

doi: 10.2313/NET-2019-06-1_01

Issue	Consequence
Inspectability	At any given point in the process, it is hard for a participating party to judge the state of the process and whether available information is up to date due to the lack of a centralized source of information.
Multiple iterations	When schedules are packed, the chance of a suggested date being available decreases. Many requests may be necessary before a valid appointment is found, increasing the chance of human error for each additional request.
Mailbox issues	The flexibility offered by email also allows for many possibilities of small, hard to trace mistakes with larger consequence. For example, a recipient might be forgotten by accident, an error that is hard to discover but can have large impact.

TABLE 1. ISSUES WITH THE CURRENT SCHEDULING PROCESS

3.1. Requirements

The new process should fulfill the following requirements to improve on problems in the current implementation.

1) **Reduced consumption of resources** is one possible goal when exchanging processes for improved versions. One resource to minimize is the participant's time, which can be measured by the average process execution time. The solution should therefore not increase this average.
2) **Runtime flexibility** is key. The improved process should prevent error prone or otherwise unwanted methods of reaching the goal while retaining desirable approaches in order to efficiently reach the target. A process that is too restrictive can face user acceptance issues, while a process that allows for too many options may be hard to maintain.
3) **Transparency and accountability** is necessary to ensure that any given task has a responsible participant. The current mail-based process is tracked by the participant's mailbox, the new process should also offer a way to view task responsibilities.
4) **Eliminating repetitive or tedious tasks** achieves increased user satisfaction and reliability. These are tasks where automation is usually viable, leading to further reduced error rates and operator strain. The feasibility of automation using BPMN machine tasks should be considered for each activity.
5) **Maintainability** is an important factor, as processes are not static and will need to be adapted over time. A solution should account for this and implement the necessary paradigms to ensure it can be maintained.

Fulfilling these requirements will improve the workflow's performance, increasing efficiency and effectivity.

4. BPMN and Camunda

The Business Process Model and Notation (BPMN) is a standardized notation for process modeling at a high level. BPMN was built to model the interaction between human and machine tasks in diagrams. We will use a BPMN toolset to implement the new solution [5]. A typical setup consists of an analyst, a modeling tool, a process engine as well as external actors and systems which interact with the process (Figure 2). One advantage of BPMN being a standardized, XML-based notation is

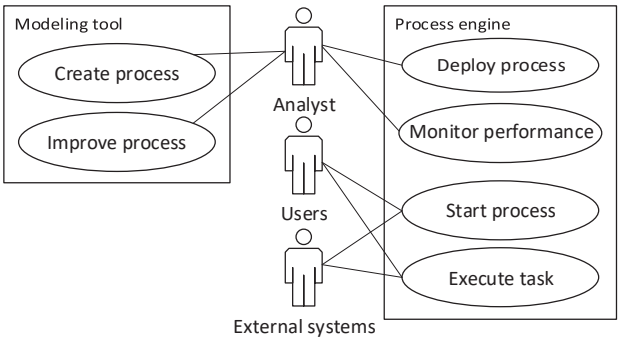

Figure 2. UML use case diagram of a generic BPMN system. An analyst maintains the process in the modeling tool and monitors it in the process engine while users and external systems interact with it.

that the format is vendor-independent. This allows components such as the modeling tool and the process engine to be chosen individually. One of these providers for a modeling tool as well as a process engine is Camunda. It was chosen due to its open source implementation as well as its suitability for fulfilling the requirements.

Three parts make up the process engine. The "Cockpit" is used for monitoring by the analysts, the task list for completing human tasks, and the administration for configuring the system. The Camunda process engine provides both user and group task lists containing activities, each of which can be assigned automatically or manually. Each activity can contain a form that allows humans to interact with data in the process. For machine tasks, the process engine can execute Java code, call REST APIs, execute scripts or simpler Java Unified Expression Language (JUEL) expressions.

The Camunda modeler is another useful part of the ecosystem, as it serves as an abstraction layer between the analyst and the XML Notation, which can be verbose. Editing a visual representation of the process will come more naturally to the analyst, especially since they might not have a technical background. The modeler also supports some Camunda specific extensions, such as forms and scripts, to simplify configuration.

5. Implementation

During the implementation, we were able to make use of some benefits provided by the BPMN and Camunda platform. However, there were also some pitfalls leading to implementation challenges.

5.1. Advantages

BPMN and Camunda are a powerful toolset for process modeling. During the implementation, the following features were found to be particularly useful.

5.1.1. Forms. A useful extension by Camunda to the BPMN specification is the ability to add forms. It allows the analyst to append HTML forms to tasks, providing a simple way to interact with the user. Variables used inside the BPMN process can be directly made available to the user for viewing or modification. There is no coding required to set up these forms since all the configuration works inside the modeler and there are no external

doi: 10.2313/NET-2019-06-1_01

systems required for providing user interaction. If the analyst decides they need something more sophisticated, it is possible to embed custom HTML forms to further improve end user experience. All the user interaction in the scheduling process, such as asking the user to specify and approve dates, was implemented using these forms.

5.1.2. Script Execution. Another useful feature offered by Camunda is that it can run scripts embedded in the BPMN diagram directly on the process engine. This allows uncomplicated interfacing with other systems and the developer is free to script parts of the system with the tools of their choice. This is a plus for automation as well, since small repetitive tasks that would otherwise be executed manually can now be scripted with a little time investment. The scheduling process used this feature for implementing logic that would have been hard to read in a BPMN diagram, such as detecting when there is no more viable date at which all parties can attend. It was also used for communicating with the email service to send notifications to the individual users.

5.2. Challenges

While the in-built functionalities included in Camunda make it suitable for many use cases, using BPMN and Camunda to implement a simple process is less straightforward than one might expect. This is illustrated by some problems encountered below.

5.2.1. Deployment. After having modeled a process in the modeler, it is time to deploy it to the process engine. This can be achieved through the modeler's user interface, but not without quirks. One problem a user might encounter is a misleading status message when trying to deploy BPMN diagrams, stating that the deployment was successful. Searching the Camunda Cockpit, it is possible that the just deployed process is nowhere to be found. This can be due to syntax or semantic errors in the BPMN diagram. Unfortunately, the user does not receive any feedback as to what went wrong and the documentation does not offer help on how to troubleshoot this problem. Instead, the only method of finding errors related to the BPMN diagram we were able to find is to search the logs of the Apache Tomcat instance in which Camunda runs. The encountered error will be listed inside a Java exception and refers to the position of the error in the BPMN XML notation. This in turn can help the user track down the error in the visual representation of the diagram to fix it in the local diagram and attempt redeployment.

While users can certainly get used to this method of error tracing, it violates some usability guidelines such as Shneidermann's "8 Golden Rules of User Interface Design" [6]. Rule two states that feedback should be instant and precise. Rule five states that errors should be prevented where possible and that help should be offered where not. Both of these rules are not well implemented, decreasing overall usability. Another problem is that the only group of users who are likely to check any log files are programmers. Since BPMN is designed to be used by both developers and business users as a way of collaborating, this solution ignores the needs of the less technically skilled users. Additionally, it is likely that users are working in a business environment where they do not have access to the logs for security reasons, which could prevent them from getting any assistance at all.

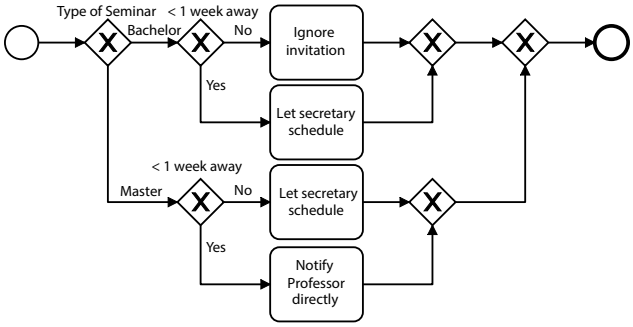

Figure 3. The BPMN equivalent of spaghetti code. Each new decision makes the number of activities double, causing the diagram to quickly increase in size. Complex decision-making should be avoided in BPMN in favor of comprehensibility, since complexity cannot be hidden by the usual object-oriented concepts such as inheritance. Instead, it is possible to externalize decisions to tools such as DMN, a notation specifically designed for decision-making [4].

5.2.2. Utilizing BPMN to the appropriate Degree. Another issue that users will encounter while using BPMN is ensuring that the level of detail which is used for process diagrams is appropriate. Too little detail will mean that changing a process to fit new requirements is inflexible, as large parts of the process are black-box activities. Too much detail, and it will become hard to interpret and modify process diagrams accordingly (see Figure 3). In this case, it would be preferable to encapsulate the decision-making in an activity and implement it somewhere else in order to hide complexity. Experience is required to be able to judge whether a given set of activities should be modeled within BPMN or whether an implementation in other tools is more appropriate.

5.2.3. Testing and Debugging. Since BPMN has similarities with programming languages, it is worth taking a look at how processes are tested and debugged within Camunda. One nice feature offered is allowing the user to view variables inside of a running process. This feature represents the basis for debugging a process, as it makes its internal state observable. Unfortunately, this is where the built-in tools for testing and debugging end. Therefore, the testing framework lacks some functionality one might expect of a programming language, such as automated testing or setting breakpoints. One particular tedious problem is when processes require a large amount of user input. Getting the process into a certain state for testing can require lots of manual interaction and this setup needs to be repeated every time a test is run. Some help is offered by the Camunda REST API, which allows the automation of human input tasks by calling appropriate API endpoints instead of entering form data manually.

6. Evaluation of Solution

In order to review the solution that was implemented, the process will now be evaluated against the previous set of requirements. A simplified view of the resulting BPMN workflow can be seen in Figure 4.

doi: 10.2313/NET-2019-06-1_01

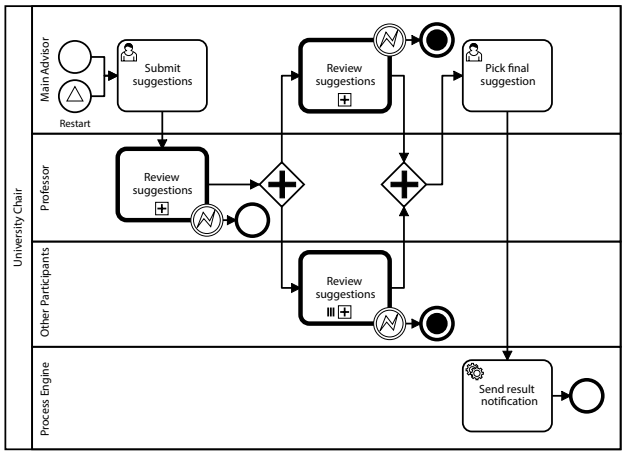

Figure 4. An extract of the final seminar scheduling BPMN process. It includes error handling and sub-processes (thick border activity) to build reusable parts of the process. Notice how the overall structure of the workflow has nevertheless stayed the same.

6.1. Review of Requirements

1) **Reduced consumption of resources** is one of the requirements specified earlier. Whether the newly implemented process is faster than the existing one depends on the probability that all participants are available for the requested appointment. A high probability causes the mail-based solution to be faster, however each retry increases the chance for error and slows down the mail process. This is where relative improvement can be observed with the BPMN-based process. Since the BPMN process is closely based upon the existing process, the number of tasks a user receives stays the same. This requirement is therefore partially fulfilled.
2) **Runtime flexibility** was maintained where appropriate. While there is a strict sequence of activities, it is possible for the advisor to cancel or restart the process at any time should the conditions change.
3) **Transparency and accountability** have been improved from the existing solution. At any given state, it is possible for participants to view the internal state of the process as well as who is responsible for the current activity. A history of previous activities is also provided. This requirement is therefore fully fulfilled.
4) **Eliminating repetitive or tedious tasks** was another previously stated goal. This was achieved by automating tasks suitable for machine processing, such as sending result notifications to participants and publishing the agreed date on the chair's website. This lessens the potential for error and increases user satisfaction.
5) **Maintainability** was a technical requirement. Some common techniques such as call activities (the BPMN equivalent of procedures and procedure calls) were used to reduce duplication.

6.2. User study

To further gain qualitative feedback on the new implementation, a small user base was asked to test the process. Some responses are shown in Table 2. It shows some things that were not considered in the design, and some potentially unwanted mechanics that were missed in testing. These provide a good basis for the analyst to continue improving the process.

Topic	Response
Communication	"The scheduler should provide a means of communicating with the others. Otherwise, I need to use email for communication in addition to this scheduler."
Disappearing tasks	"Sometimes tasks disappear from the task list without an apparent reason." (Later, it was found this happens when other people submit their review in parallel and there are no more viable suggestions)
Information flow	"It does a good job tracking what needs to be done with multiple running requests."

TABLE 2. USER'S FEEDBACK AFTER TESTING THE BPMN PROCESS.

The newly introduced process is therefore a viable alternative to the e-mail based solution. While there are other known solutions to the scheduling problem, this case study was useful to examine some benefits and drawbacks of BPMN based process modeling in relation to academic processes.

7. Conclusion

Processes are part of many systems and managing them can help improve their performance. The effects of using process modeling were examined using a case study of a process at the university chair. Some problems with the current process and requirements for an improved version were determined. A new solution was then implemented using BPMN and Camunda, which was later evaluated against the previous requirements. We showed that both the email-based solution and the BPMN-based solution each have their advantages and disadvantages. While the email-based solution is less maintenance intensive, its complexity and error rate increase with a decreased probability of all participants being available at a given request. On the other hand, while the BPMN implementation offered good accountability, it reduced the overall flexibility. With a small user study, some additional feedback was collected and some minor issues found. Overall, the new solution was deemed a viable alternative. When applied to the right processes with the right requirements, BPMN and Camunda can help achieve significant improvement in efficiency and effectivity.

References

[1] Merriam-Webster, "Dictionary", https://www.merriam-webster.com/dictionary/. Accessed 25 Sept. 2018.

[2] camunda services GmbH, "BPMN 2.0 Symbol Reference", https://camunda.com/bpmn/reference/. Accessed 02 Sept. 2018.

[3] camunda services GmbH, "Best Practices for creating BPMN 2.0 process diagrams", https://camunda.com/bpmn/examples/. Accessed 25 Sept. 2018.

[4] camunda services GmbH, "DMN Tutorial", https://camunda.com/dmn/. Accessed 25 Sept. 2018.

[5] Object Management Group (OMG), "Business Process Model and Notation (BPMN)", https://www.omg.org/spec/BPMN/2.0.2/PDF, December 2013

[6] Dr. O. N. N. Fernando and Dr. Y. He, "Human Computer Interaction", NTU School of Computer Science and Engineering, January 2018

doi: 10.2313/NET-2019-06-1_01

Performance of Secure Multiparty Computation

Ludwig Dickmanns, Marcel von Maltitz*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ludwig.dickmanns@outlook.de, vonmaltitz@net.in.tum.de

***Abstract*—With the recent advancements in modern computer technology secure multiparty computation (SMC) evolved from a mere theoretical approach to a number of actively developed software projects. A major advantage of SMC is that it allows a set of parties to jointly calculate a function without any party revealing it's input.**
In our study we evaluated the influence of network parameters on the performance of a SMC framework, in order to derive an outlook for the feasibility of SMC applications in the future. For the evaluation the following parameters and the corresponding measurements were chosen: The impact of increasing the number of peers on execution time and protocol invocations and the effect of added network latency and decreased bandwidth on the execution time.
Our results indicate that SMC is a feasible option, especially in setups with a high bandwidth, low network latency and a limited number of peers. Linear increase in peers led to a linear increase in execution time and protocol invocations. The execution time increased drastically for a transmission rate of 10 MBit/s or lower. However, added network latency had the most significant negative impact.

***Index Terms*—secure multiparty computation, performance, measurement**

1. Introduction

The main goal of Secure Multiparty Computation is to allow several parties calculating a joint function. The corresponding inputs of each party are kept private and no Trusted Third Party should be required for the calculation.

The most prevalent example for SMC is Yao's Millionaires' Problem: Two millionaires wish to determine whom of both is wealthier – without either one of them revealing their credit balance. As mentioned in the beginning, no Trusted Third Party should be required. In his paper "Protocols for Secure Computation" from 1982, aforementioned A. C. Yao proposes a solution to this problem, which satisfies the above mentioned criteria. Furthermore, the researcher describes a generalized approach for similar problems with more than two parties calculating a collective function without revealing their inputs, e.g. "Mental Poker" [1]. However, at this point in time a practical implementation was not feasible due to lack in computational power. Fortunately – with the advancement of technology over the recent years – computers are now capable of performing such tasks in an appropriate amount of time and thus SMC-Frameworks are emerging [2] [3].

Besides computational power there is another important point to consider when answering the feasibility question: Network performance. The purpose of this paper is to investigate in and help answering the following question: How do network parameters influence the performance of SMC? Hence, measurements were carried out examining the influence of the following parameter:

- Number of peers
- Network latency
- Transmission rate

Only for the first one, number of peers, we identified the impact on the amount of evaluated protocols, because the other two parameters are not influencing it. For all three of them execution time measurements were carried out. This is the most important factor in the context of usability because the application should be able to operate in an user-acceptable amount of time.

The remainder of this paper is structured as follows: After identifying related work in Section 2, there will be information about our testing setup in Section 3 divided in two parts: Testbed (3.1) and the SMC framework of choice and our adjustments to it (3.2). Then, the results of the measurements are shown in Section 4 and in Section 5 those results and the corresponding consequences are discussed. In the last Section (6) we draw a conclusion regarding our results and provide an outlook for future research.

2. Related Work

Firstly – as discussed in the introduction – the theoretical foundation for SMC was layed out in 1982 by A. C. Yao [1]. Secondly several SMC frameworks were implemented. Hence, the third step is to evaluate the performance of such applications in order to discuss whether the technology is applicable.

One publication investigating on this topic is "A performance and resource consumption assessment of secure multiparty computation" by Marcel von Maltitz and Georg Carle [4]. Here, the researchers analyzed the following parameters:

- Number of peers
- Network latency
- Transmission rate
- Packet loss

doi: 10.2313/NET-2019-06-1_02

- Input data parallelization

With the purpose of examining data about their impact on the following performance indicators:

- Execution time
- CPU cycles
- Heap memory consumption
- Transmitted bytes

Their research work resulted in a promising outlook for SMC in the future. Intranet applications can already be considered feasible, however for Internet and mobile Internet applications the main bottleneck is network latency. A difference to this paper is the SMC protocol (BGW) used by the software under test. BGW will not be explained in this paper, however, in Section 3.2.1 there is a short introduction to SPDZ, which is the SMC protocol used by the software we tested.

In this paper there will be an analysis of three of the above five parameters. We used the same network setup in order to either validate the thesis by delivering a similar outcome or providing room for discussion by showing different results. However, we did not analyze packet loss and input data parallelization as this would exceed the frame of this study. Furthermore, we did not investigate CPU cycles, heap memory consumption and transmitted bytes, for the same reason.

Further measurements were taken in [5] [6].

3. Test Setup

The setup, in which the time measurements were carried out, consists of two parts. The first of which is the testbed, i.e. the hardware the test and measurement software was ran on. The second part is the SMC framework software – namely FRESCO – and our adjustments to it. Firstly, general information on FRESCO is given. Then, the tested application is introduced. Afterwards, we explain how the measurements were taken.

3.1. Testbed

For our tests we used a range of three to 17 physical peers in order to derive the influence factor of adding additional peers to the network. The hardware for each host was equal: A four core Intel Xeon CPU running at 2.50GHz with 8192KB of cache and 16GB RAM. All hosts were connected to each other with an 1 GBit networking interface and the default link latency is around 0.18ms. The network is organized in a star topology: A central switch in the middle is connected to three other switches. The ladder are connected to five to six hosts. As the operating system of choice on each machine Debian Stretch (9.4) was used with a 4.9 Linux kernel.

3.2. FRESCO

In this subsection we introduce the secure multiparty computation framework, which we used for our tests. Then software under test is introduced. Here, we adjusted an demo application, which is part of the framework, and those adjustments are explained. Afterwards, the methodology for the measurements is introduced.

3.2.1. General. The FRamework for Efficient Secure COmputation (FRESCO) is developed by the Alexandra Institute in Denmark. It is licensed under the open source MIT license. According to their documentation [7], the framework is already prototypically in usage, e.g. to evaluate surveys without revealing the answers of the participants. The code is written in Java, which helps with platform independence. Summarizing, the main goals of FRESCO are providing an infrastructure for uncomplicated SMC application development, with an easily adjustable design. Earlier mentioned protocols are defined in a protocol suite and represent the base functions of it. Hence, applications are built using protocols. Batches contain a certain number of protocols, which are evaluated in parallel. In addition, the effort for developing an individual protocol suite is reduced by contributing a framework of reusable patters. A central feature of FRESCO is an extra abstraction layer, in order to separate algorithm/application development from the mathematical realization of the underlying SMC protocol. Finally, FRESCO tackles scalability issues by supporting preprocessing and parallel execution. FRESCO is actively developed and maintained. The SMC protocol used by FRESCO is SPDZ, which allows secure arithmetic calculations for multiple parties via secret sharing [8]. A great example to explain and illustrate arithmetic operations with secret sharing is the addition with multiple parties. In this case, n parties wish to collaboratively calculate the sum of their inputs, here integer values. Party i contributes the input x^i. Each party splits it's inputs into n randomly sized parts $(x_1^i, ..., x_n^i)$ – so called shares – in a way that adding up all of the shares results in the input (x^i). E.g. for the input x^i this means $\sum_j x_j^i = x^i$. In the next step, the shares are exchanged between the parties: Party j receives share j of each party $(x_j^1, ..., x_j^n)$. Then, Party j calculates partial sum r_j of the received shares. In the last step all partial results are exchanged allowing each party to calculate the total sum [9]. The previously cited article and its follow-ups provide further insights to arithmetic operations with secret sharing and SPDZ.

3.2.2. Tested Application. For our studies we used version 1.1.2 of FRESCO [2], which contains demo applications. In one of those three parties are collaboratively calculating the sum of an integer array, which is the input of party one. Parties two and three initially had no inputs. In order to make this example more realistic, however, we adjusted it in order to allow more than three parties, with each of them contributing their own input array. Instead of the sum, the application calculates the mean and the variance of all of the inputs. Previously the input was hard coded inside the FRESCO code, but in our example the inputs are now read in from a CSV file. With those adjustments the example was made more versatile for testing as this allows to draw a conclusion about the impact of adding extra peers.

3.2.3. Measurements. As mentioned in the beginning of this paper, measurements were taken on the evaluated protocols and the corresponding batches and execution time. When running the FRESCO application, each peer produces an individual log file, where data about the last run are stored. Both evaluated performance indicators

were extracted from those files and the collected data is illustrated in the figures of this paper in Section 4. Each measurement was carried out ten times and the median of those measurements was used for the illustrations.

4. Results

Separated by the input parameters – namely number of peers (4.1), network latency (4.2) and transmission rate (4.3) – this section addresses the results of this study. Hence, interpreting the impact of the above parameters on the execution time and the amount of transferred data is the content of this chapter.

4.1. Number of Peers

For our tests we used a range of three to 17 peers in order to investigate in the change in execution time (4.1.1) and protocol invocations (4.1.2) with an increasing amount of peers.

4.1.1. Execution Time. The effect of an increasing number of peers on the execution time is demonstrated in Figure 1. The x axis shows the number of peers, whereas on the y axis the execution time can be seen. A higher amount of peers leads to more input for computation as well as an increase in communication between the peers. Both of these factors result in an extended execution time. In addition, the figure illustrates a linear behavior, which can be considered positive as e.g. quadratic behavior would be much worse.

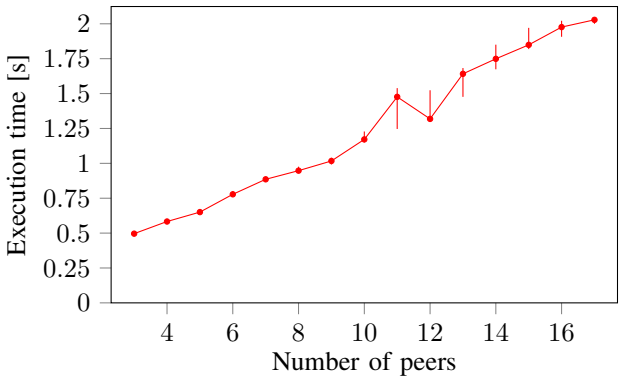

Figure 1. The execution time depending on the number of participating peers.

4.1.2. Protocols and Batches. Besides the execution time, the amount of protocol invocations and the number of corresponding batches was measured. As shown in Figure 2 an increase in peers led to more protocol invocations in an higher amount of batches. Both measured indicators seem to follow linear behavior – as the execution time does – in relation to the number of peers. While the quantity of batches grows relatively slowly, the number of protocol invocations increased faster in comparison. However, as both numbers are increasing linearly, this example provides an argument in favor of the feasibility question. For very large applications this still remains a point to consider, but even here (and from the execution time standpoint) linear increase is by far more acceptable in contrast to e.g. quadratic or cubic behavior.

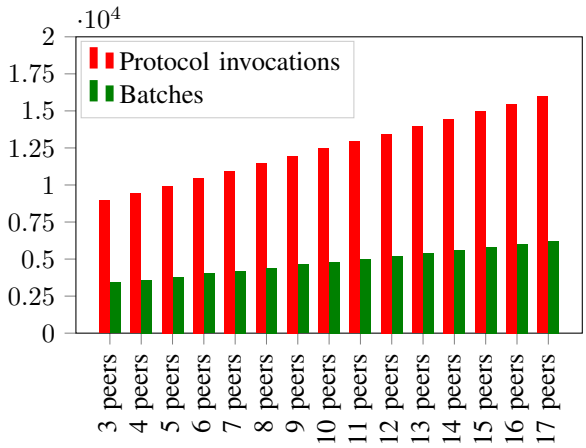

Figure 2. The amount of native protocol invocations and the number of batches in which they have been executed depending on the number of participating peers.

4.2. Network Latency

Another parameter evaluated in the study for this paper is network latency and how it affects the execution time of the application. In order to add latency `tc` was used. Figure 3 illustrates how additional network latency affects the execution time for the range of three to 17 peers connected to each other with a bandwidth of 1000 MBit/s. The increase in execution time in relation to network latency behaves similar to a root function. However, the growth in run time increases significantly. As an example, the execution time without additional latency is approximately under five seconds for the range of three to 17 peers, whereas a delay of 10ms results in an factor five to 17 growth in execution time for the corresponding number of peers. After this strong increase the slope flattens. Increasing added network latency from 10ms to 50ms (factor five) results in an approximately factor two increase in execution time.
Inferentially, it has to be stated that the network latency can result in a problem for SMC. Especially for mobile and Internet use cases this imposes a problem and possible limitations as in those cases a latency of 10ms is already quite optimistic. Hence, for such applications, latency can lead to an significant increase in execution time.

4.3. Transmission Rate

The last parameter we analyzed for this study was the transmission rate. In order to decrease the bandwidth from a maximum of 1000 MBit/s to a minimum of 1 MBit/s the same tool as in Section 4.2 was used (`tc`). Equal to the other measurements, the experiments were made with three to 17 hosts. Execution time was measured in the aforementioned range for 1 MBit/s, 10 MBit/s, 100 MBit/s and 1000 MBit/s. Figure 4 illustrates that there is no significant change in execution time between 1000

doi: 10.2313/NET-2019-06-1_02

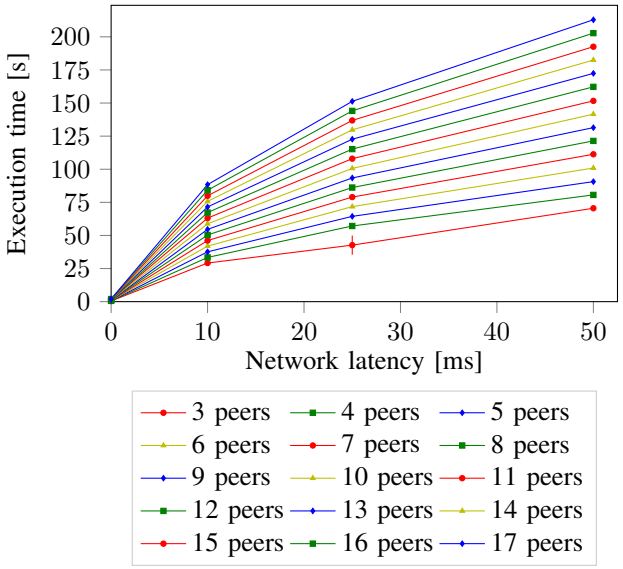

Figure 3. The execution time of the secure computation depending on the number of peers and on the network latency of the network. The latency highly influences computation time irrespective of the available transmission rate.

MBit/s and 100 MBit/s. However, from 100 MBit/s to 10 MBit/s there is a slight increase in time for execution. In the last interval, from 10 MBit/s to 1 MBit/s, the increase in execution time is much steeper, especially when the lower bandwidth is combined with a large number of participating peers. This behavior can be interpreted as follows: With 10 MBit/s and lower the transmission rate constitutes a bottleneck, but as soon as this threshold is exceeded there are only slight improvements in execution time.
Therefore, this may result in problems for large scale applications with a huge amount of hosts and some peers having lower bandwidths, e.g. mobile apps and Internet applications. However, for smaller networks with a high transmission rate between the peers, for example an intranet setup, the SMC approach remains more feasible.

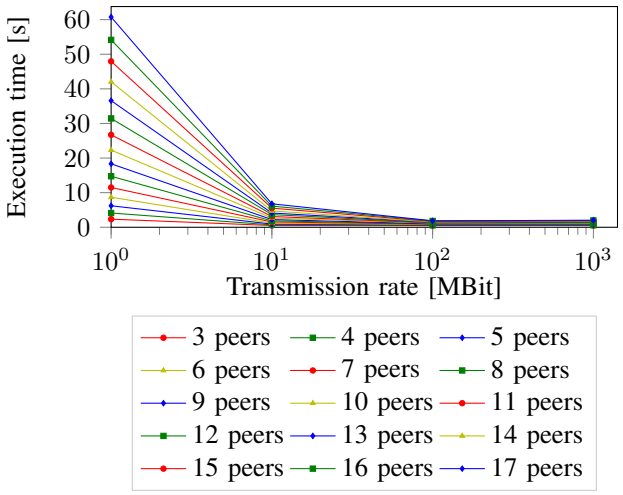

Figure 4. The execution time of the secure computation depending on the transmission rate. The shown case has no additional network latency.

5. Discussion

The purpose of this Section is to put the results from Section 4 in context and – as mentioned in Section 2 – compare them to the results of "A performance and resource consumption assessment of secure multiparty computation" [4].
As discussed in Section 4.1, the execution time increases linearly with an increasing number of peers in our case. The previously mentioned study came to the same results, which strengthens the thesis of having linear behavior.
For protocol invocations no comparable measurements were taken. In our case, we were able to also identify linear correlation between number of peers and protocol invocations.
Transmission rate can be considered a smaller problem. Both studies came to the result that a relatively low bandwidth of 1 MBit/s significantly increases the execution time, but even an increase to 10 MBit/s reduces time of execution drastically. Further increase in transmission rate only has a small effect.
In our study network latency constitutes the most important factor, as it had the largest absolute impact on the execution time. The related study supplies equal results, which strengthens this thesis.

Hence, for intranet applications, which are mostly connected with broad and fast connections, secure multiparty computation can be considered a viable option. Under the described circumstances, it is possible to keep execution time acceptably low. However, for applications with a less powerful network infrastructure, for example mobile apps, this can result in a high execution time, as the factors combine, i.e. a low transmission rate (EDGE: 384 KBit/s, 3G: 7.2 MBit/s (HSPA) [10]) and a higher network latency(EDGE: 200-450ms [11], 3G: 100-350ms [12]).

6. Conclusion

Summarizing, SMC provides a great software solution when several parties wish to collaboratively calculate a function, without either of them revealing their input. The theoretical concept already existed over 35 years ago, but recent advancements in computer and network technology allow practical implementations. However, there are still considerable limitations to secure multiparty computation. With state-of-the art technology and networks, intranet applications can already be considered feasible due to a limited amount of peers, low network latency and a high transmission rate. In such setups an acceptable execution time seems realistic. However, in use cases with less optimal circumstances there is a significant increase in execution time. While a decrease in transmission rate leads to an increase of up to approximately factor five, the influence of the amount of peers was at factor three from three to 17 participants. The greatest absolute increase in execution time came from the network latency. Even a small delay of 50ms increased the time of execution with up to a factor of over 100. With those limitations real-time or close to real-time applications are unfeasible at the moment. However, for use cases with softer time restrictions or faster infrastructure, secure multiparty computation can be feasible. With further advances in internet and network

technology – increasing bandwidth in combination with lower network latency – SMC can exceed the limitations of today. Hence, it remains an interesting topic for further research and investigation.

References

[1] A. C. Yao. Protocols for Secure Computations. *Proceedings of the 23nd Annual Symposium of Foundations of Computer Science*, Washington, DC, USA: IEEE, pp. 1-5, 1982.

[2] A FRamework for Efficient Secure COmputation. https://www.github.com/aicis/fresco. 2018.

[3] Sharemind MPC. https://sharemind.cyber.ee/sharemind-mpc/.

[4] Marcel von Maltitz and Georg Carle. A performance and resource consumption assessment of secure multiparty computation. *CoRR*, abs/1804.03548, 2018.

[5] M. Burkhart, M. Strasser, D. Many and X. Dimitropoulos. SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics. *Proceedings of the 19th USENIX Conference on Security*, p. 15, 2010.

[6] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. *IACR Cryptology ePrint Archive*. Springer, 2008, no. October, p 289.

[7] FRESCO Documentation. https://fresco.readthedocs.io. 2018.

[8] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. *Lecture Notes in Computer Science*, vol. 7417, 2012, pp. 643–662.

[9] Bristol Cryptography Blog. https://bristolcrypto.blogspot.com/2016/10/what-is-spdz-part-1-mpc-circuit.html. 2016.

[10] Mobiles Internet. https://de.wikipedia.org/wiki/Mobiles_Internet. 2018.

[11] Alles zum Thema edge. https://www.onlinekosten.de/mobiles-internet/edge/.

[12] 3G/4G Ping Times/Latency. https://www.evdoinfo.com/content/view/4818/64/.

Overview of TCP Congestion Control Algorithms

Moritz Geist, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: moritz.geist@tum.de, jaeger@net.in.tum.de

***Abstract*—The current set of congestion control algorithms is split into three primary groups regarding their function and can easily be categorized and therefore characterized. This paper compares these different classes in compliance with their respective advantages and disadvantages. Each algorithm on its own is well researched, however, it is difficult to predict how they perform when used together in the Internet due to the unpredictable behaviour of it.**

***Index Terms*—tcp, congestion control algorithm, measurement, high-speed networks**

1. Introduction

With the world being more and more connected through the Internet, the underlying network has to work increasingly efficient to achieve a high-performing and stable connection all the time. That is a very difficult goal to achieve without the help of the transmitting computers. If every computer connected to the Internet would just send packets as fast as possible, the slowest links or most utilized networks would get overloaded to the point of routers dropping packets instead of passing them on to the next node. This leads to a severe performance hit for Internet applications due to the whole stream having to wait for the packet to be retransmitted. To avoid this and keep a high throughput, while not losing packets, computer scientists have come up with several congestion avoidance algorithms. These algorithms work by controlling the size of the congestion window. The congestion window limits the number of packets that can be inflight, meaning waiting for acknowledgement, at any time and is created for every TCP connection. As long as there are less packets inflight than the size of the congestion window, new packets will be sent out and transmitted. The packets get removed from the congestion window as soon as they are acknowledged, which allows for the next packet to be sent. By this definition, the optimal congestion window size is equal to the Bandwith Delay Product (BDP). The BDP is calculated using $BDP = bandwith \cdot RTT$. The bandwith to be used is the total usable bandwith of the slowest link on the path, and RTT is the total round trip time of the stream. This is due to the nature of routers between the sender and the receiver: Internally, they operate a packet queue that incoming packets will be appended on. The packet at the head of the queue will be processed and transmitted to the next node. If it happens that more packets come in than the number of packets that get sent out, the packet queue will grow until a certain limit. If the limit is reached, packets will be dropped until there is new space in the queue. If the sender just continues to send out packets as fast as possible, all packets would get dropped on that link. Instead, the congestion control algorithm reduces the congestion windows size until no more packets are lost on their way. The result of a higher congestion window than BDP is therefore a queue filling up at the slowest link, while a smaller congestion window decreases the size of the queue gradually.
The primary difference in congestion avoidance algorithms is how they detect an overloaded link in between them and how they increase and decrease the congestion window. This paper will compare different methods of detecting and handling the size based on some existing algorithms.
This paper will first explain the different techniques on how a congestion control algorithm can operate in Section 2, followed by examples that highlight each of the different approaches in Section 3. In Section 4, the algorithms will be discussed in how they perform compared to each other and especially while active as parallel streams in a network.

2. Background

In this section, we compare how different congestion avoidance algorithms detect an overloaded link on their path. While it is most common to utilize only one of these three strategies, some algorithms use combinations of them. As Lefteris Mamatas et al. specify in [1], the algorithms can be classified into three distinct groups: black box algorithms that do not rely on any state information about the network and only rely on binary feedback; gray box algorithms that do active measurements to "estimate available bandwidth, level of contention or even the temporary characteristics of congestion" [1]. Third, green box algorithms do have exact knowledge about the state of every (or most) part of the network, either by being implemented in every part of it or every link reporting its status to the sender.

2.1. Loss-based Algorithms

One of the most common ways of detecting an overloaded link is by reacting to packet loss. This is considered a black box method, as it only relies on the binary input of a packet being lost or not. In case a packet is not acknowledged after a certain time, called the retransmission timeout, or after receiving three duplicate ACKs [2], it is considered missing and will be retransmitted. This also indicates to the congestion avoidance algorithm to

doi: 10.2313/NET-2019-06-1_03

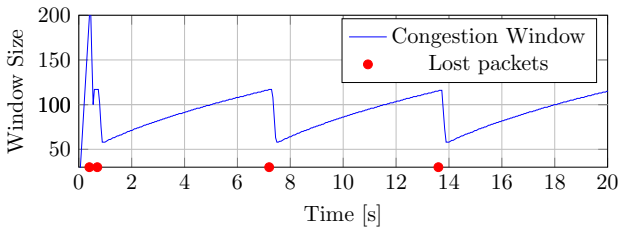

Figure 1. Congestion window resize after a lost packet in TCP Reno

shrink the congestion window, so less packets are sent out. In the most simple algorithm, called TCP Reno (seen in Figure 1), the congestion control begins with a phase called "slow-start" up to $t = 1s$ [3]. In that, it the window size is increased by one for each acknowledged packet resulting in an exponential growth, until the first lost packet is registered. After that, the congestion window is halved on loss, and will be increased steadily, but in a linear way. This results in a sawtooth-like graph for the congestion window size like in Figure 1. Loss-based congestion avoidance algorithms only need to be implemented by the sender, making them exceptionally easy to deploy. The receiver, who also is a sender by at least sending out acknowledgements, can use a totally different congestion avoidance algorithm. The problem with these algorithms is that packets can get lost for reasons other than an overloaded link, for example, if an actually broken, or a less reliable link is used on the way. This results not only in a bad performance due to retransmits, also the congestion window is decreased without any need making it smaller than the BDP and therefore decrease effective throughput. Even if packets are only lost due to overloaded links, these algorithms can never be perfect: The window size is increased as long as there is no packet loss, which in turn means that there always will be at least one lost packet every so often, as seen in Figure 1. This can lead to problems with applications that require near-real-time communications, like Voice-Over-Ip or online games. Algorithms based on packet loss are therefore the easiest to implement, while also theoretically most limited considering accurate functionality.

2.2. Delay-based Algorithms

Another way of detecting an overloaded link is by measuring the delay in which acknowledgements arrive, also known as the round trip time (RTT). By the criterion listed in Section 2, these algorithms are grey box algorithms, as they use more advanced measurements to examine and monitor the status of the network. Especially by monitoring changes in RTT, these algorithms can react to a congestion earlier than purely loss-based algorithms, as most of the time the RTT increases gradually before a packet is actually lost. The queue fills up like explained in Section 1 when the congestion window is bigger than the BDP. A nonempty queue means that packets have to wait in line to be sent on, leading to a higher RTT. Only when the queue size reaches he maximum buffer size, packets are dropped instead of being queued. This has the following advantages: Delay-based algorithms can react sooner to congestion, maybe even before the first packet is lost at all, which can have a positive impact on the performance of some applications. Also, instead of having to cut down the congestion window in half, it can gradually shrink the congestion window relative to measured increase in RTT. This can lead to an improvement in throughput compared to other black box algorithms in isolated environments. A very basic implementation of a delay-based algorithm is TCP Vegas [4]. An example of this behaviour can be seen in Figure 2. As soon as the network gets congested at *t=10*, the buffer of the overloaded link starts to fill up, resulting in the average RTT increasing. The sender reduces the congestion window to prevent any packet losses.

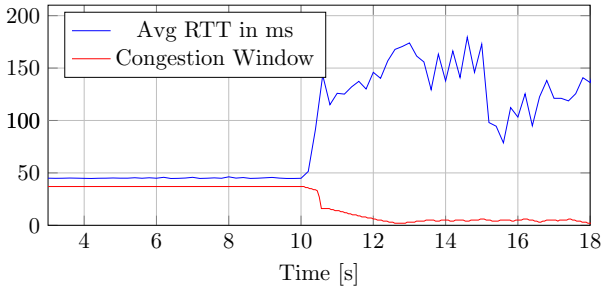

Figure 2. Decrease of Congestion Window Size

While a delay-based algorithm is alone in a network, it is able to adjust its congestion window to a perfect size: Maximum throughput combined with no packet loss and minimum RTT. It gets difficult as soon as another connection uses the same link that is more aggressive, like a loss-based algorithm mentioned in Section 2.1. As these will enforce one or more lost packets from time to time, this affects the delay-based stream as well because of the increased queueing delay, leading to it reducing the congestion window to an absolute minimum. Also, delay-based algorithms can react to loss similar to loss-based algorithms as well, as seen in Section 3.2. How the detection technologies work together will be described in Section 4.

2.3. Signal-based algorithms

As the definition in [1] states, with green boxes "the network communicates its state to the transport layer". This is achieved through signal bits, with which an overloaded link notifies the sender of its state. The actual implementation of this can vary. The TCP protocol header contains six unused bits that are reserved for future use as well as an option field that can carry more complex information [5].The advantage of this algorithms is obvious: With complete knowledge of the status of the network, it becomes a matter of algorithmic design of how the sender should adjust its congestion windows size. It is even possible to prioritize different streams of data without exceeding the load on a link, making sure it does never get actually overloaded and incur high delays or drop packets. A shortcoming of this method is illustrated in Figure 3. The link between the router and the receiver is slower than the link between the sender and the router, which means it will be overloaded at some point. The router detects this as soon as the internal buffer starts

doi: 10.2313/NET-2019-06-1_03

to fill up, and will set signal-bits or options in the next packet it forwards to the receiver accordingly. The receiver will then acknowledge the packet and adopt the extra information in the corresponding acknowledge-packet.

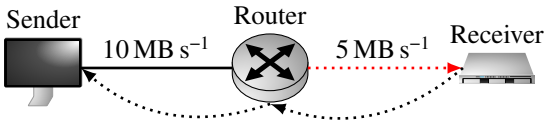

Figure 3. Path of the signal

As visible in Figure 3, the information about the increased load needs to travel at least three hops, depending on the network layout. It follows the dotted lines back to the sender. Therefore, the information takes at least half of one round trip time to reach the sender. The big disadvantage of signal-based algorithms or green box algorithms is that they need to be deployed on all parts of the network, including the receiver and every router. This makes it practically impossible to use them in the open Internet, so they are only useful in closed networks like offices or data centers where each of the network components is controlled by a single instance. Loss-based 2.1 and delay-based 2.2 algorithms do not require this.

3. Implementation

This section will showcase some more- and less popular congestion avoidance algorithms and highlight their benefits and shortcomings. It will cover all of the before mentioned types of detection from Section 2.

3.1. TCP Cubic

TCP Cubic is the current default congestion avoidance algorithm in the current Linux kernel, which makes it one of the most used algorithms. It has been developed by Sangtae Ha et al. in 2008 and the specification is made available here [6].
TCP Cubic is completely loss-based and therefore a black box algorithm. It can be viewed as an improvement over TCP Reno described in Section 2.1 and Figure 1. After the same slow-start-phase that is present in TCP Reno it behaves similarly when receiving a lost packet. The main and only difference is how it increases the congestion window. Instead of increasing it with a constant (or at least linear depending on the current RTT) function it splits the increase function into two phases that are illustrated in Figure 4: First, it will increase the congestion window based on a concave cubic function (right curved) until it reaches a value called W_max at $t = 8$. This is the size the congestion window had when the last congestion event (packet loss) occurred. After reaching that point again, the function begins to increase slowly at first, but with increasing speed later on (convex, left curved). This method of growth causes the congestion window and therefore sending rate to stay close to the last known highest value as long as possible, while still be able to go beyond that. Consequently, TCP Cubic behaves very similar to TCP Reno while being able to faster recover from a loss and slower run into the next one.

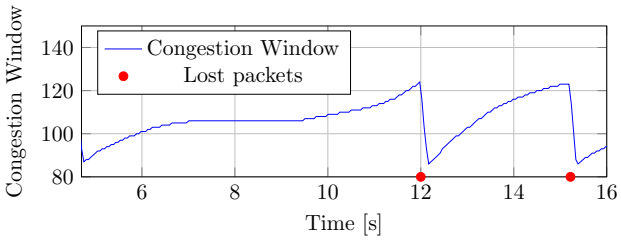

Figure 4. TCP Cubic Congestion Window Size

3.2. LEDBAT

Low Extra Delay Background Transport (LEDBAT) is a delay-based congestion control algorithm that has been technically documented in 2012 [7]. It can be seen as a grey box algorithm, as it measures the time a packet travels from sender to receiver, not just the round trip time. To do this, a timestamp is appended to every outgoing packet that the receiver then subtracts from his local time and responds the one-way delay to the original sender using the acknowledgement packet. The sending side then considers the difference in delay over time, so clocks do not need to be synchronized. LEDBAT purposefully only uses the one-way delay for its calculations, as it is designed for primarily one way bulk-transfer applications like file-sharing and software updates. Its goal is to reduce the impact on contending other streams while still being as fast as possible. As seen in [7], LEDBAT is, in general, less aggressive than TCP Reno or Cubic due to it also halving the congestion window on loss in addition to shrinking with increasing delay. The ramp-up function is never faster than TCP Reno, therefore it will not interfere too much with other traffic. As a result, LEDBAT is very useful for data transfers that do not require real-time information processing but instead just need to get done in some amount of time. Currently LEDBAT is used by the BitTorrent system [8] as well as for updates in some operating systems.

3.3. TCP Westwood

TCP Westwood (and TCP Westwood+) is a combination of a loss- and delay-based congestion control algorithm [9]. They have been developed to improve efficiency in paths with a large bandwith-delay product and a potential packet loss such as long wireless links. In general, it behaves similarly to TCP Reno in that it features equal slow-start and congestion avoidance phases. The difference is how it handles congestion events or packet losses. Instead of halving the congestion window, Westwood uses an algorithm to estimate the real end-to-end bandwidth with which it then adaptively sets the slow-start threshold and congestion window. Due to the bandwidth esitamtion Westwood works fine in a network with more streams, unlike TCP Vegas as will be explained in Section 4. TCP Westwood+ works similar, but employs a different algorithm to estimate bandwidth, as the original algorithm was found to be flawed [10] due to compression of acknowledgement packets.

doi: 10.2313/NET-2019-06-1_03

3.4. TCP MaxNet

TCP MaxNet is a green-box algorithm from 2002 that features active feedback about the status of the network to the sender [11], [12], [13]. The routers on the way each calculate a price using an active queue management algorithm (AQM). These calculations factor in a demand function, the next links capacity and utilization. Given equal demand functions, there is max-min fairness [11]. By scaling this function, it is possible to prioitize a stream (weighted fairness). The calculated price is then transported back to the sender along the links using the max() Function 1 with p_i being the individual price value of the router at position i.

$$price = \max_{p_i} \tag{1}$$

A detailed definition of the price can be seen in [13]. The client will then adjust its congestion window according to the determined price, so the effective sending rate adjusts to the slowest link in the network. It therefore is easy to define and predict how the network behaves, as long as there are no competing streams that do not use TCP MaxNet.

4. Evaluation

An important consideration when developing a new congestion control algorithm is how it behaves in real-world scenarios, where overloaded links are hit with different streams at once, and therefore with several different rates, and different congestion avoidance mechanisms. For example, a delay-based stream will reduce its congestion window way before a competing loss-based stream observes a lost packet. An example of this is seen in Figure 2, where the congestion window of a TCP Vegas stream is visible. TCP Vegas is solely delay-based and shrinks the congestion window when the RTT increases [14]. The effect can be seen by also measuring the throughput at the congested link, which is plotted in Figure 5, showing the same timeframe as in Figure 2. As soon as the Reno stream starts at $t = 10$, it immediately congests the link in the slow-start phase, to which TCP Vegas reacts with reducing the congestion window drastically. Reno will then keep the link congested regularly with a congestion window similar to Figure 1, leading to TCP Vegas never increasing its congestion window again.

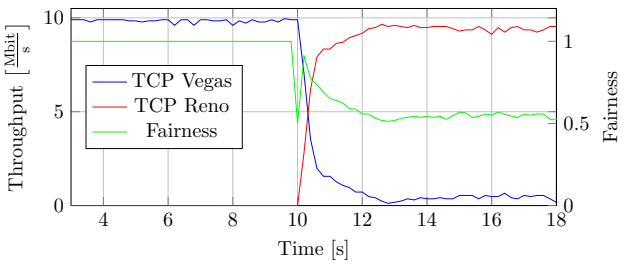

Figure 5. Throughput at the congested link

This can be calculated using a fairness measurement, a value that describes how well the total throughput is shared among all streams. One way to calculate such a value is by using an equation developed by Raj Jain [15]. It produces values ranging from $\frac{1}{n}$ (not fair), where n is the number of streams, to 1 (fair). For the above example, the index is of value 1 until $t = 10$, and then drops to about 0.5. Therefore, this example showcases a worst-case scenario in terms of fairness. Other delay-based algorithms try to improve on that shortcoming, for example, Westwood and Westwood+ behave better in the same scenario with fairness values of around 0.8 on average, while still being outperformed by TCP Reno in terms of effective throughput. Generally, tt would be best if everyone would be using the same algorithm, just like a green-box algorithm like MaxNet (see Section 3.4) encourages. These do then accomplish MaxMin fairness, meaning equal share for every stream with a fully saturated but not overloaded link.

5. Related Work

The idea of increasing the overall efficiency of the Internet is quite interesting, therefore a lot of people already tried their best at developing a new, best congestion control algorithm to the point that a "Yet Another Highspeed TCP"-algorithm "YeAH-TCP" [16] exists. Other newly introduced alforithms include BBR developed by Neal Cardwell et al. that tries to determine the actual level of congestion using active probing of the network [17] and PCC Vivace by Mo Dong et al that utilizes machine learning to improve network efficiency [18]. There is also research that compares the performance of algorithms like L. Grieco does in [19]. The paper *The macroscopic behavior of the TCP congestion avoidance algorithm.* [20] explains how TCP congestions can affect the real Internet.

6. Conclusion and Future Work

While there are many congestion control algorithms that each have their pros and cons, it is very difficult to have them existing next to each other, unless it is the design goal (like with LEDBAT in section 3.2) to have a lower-priority stream. The existing methods for judging fairness work well to calculate the fairness afterwards, but with the chaotic nature of the Internet it is very difficult to predict the behaviuor of the packets. All the algorithms more or less expect that every packet travels through the same links every time, which is not guaranteed in any way. There are even projects that try to increase throughput and failure handling by explicitly using different interfaces and links like MultiPath TCP [21]. In future work, it would be interesting to look at the performance if algorithms in changing network environments and how fast they adapt to big variations in maximum throughput and RTT.

References

[1] L. Mamatas, T. Harks, and V. Tsaoussidis, “Approaches to congestion control in packet networks,” *Journal of Internet Engineering*, vol. 1, no. 1, 2007.

[2] W. R. Stevens, “Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms,” 1997.

[3] M. Allman, V. Paxson, and E. Blanton, “Tcp congestion control,” Tech. Rep., 2009.

[4] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, vol. 24, no. 4.

doi: 10.2313/NET-2019-06-1_03

[5] P. SPECIFICATION, "Transmission control protocol," 1981.

[6] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[7] M. Kuehlewind, G. Hazel, S. Shalunov, and J. Iyengar, "Low extra delay background transport (ledbat)," 2012.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: the new bittorrent congestion control protocol," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*. IEEE, 2010, pp. 1–6.

[9] L. Grieco and S. Mascolo, "End-to-end bandwidth estimation algorithms for westwood tcp congestion control," in *Information Technology Interfaces, 2003. ITI 2003. Proceedings of the 25th International Conference on*. IEEE, 2003, pp. 563–568.

[10] R. Ferorelli, L. A. Grieco, S. Mascolo, G. Piscitelli, and P. Camarda, "Live internet measurements using westwood+ tcp congestion control," in *GLOBECOM*, vol. 2, 2002, pp. 2583–2587.

[11] B. Wydrowski and M. Zukerman, "Maxnet: a congestion control architecture," *IEEE Communications Letters*, vol. 6, no. 11, pp. 512–514, 2002.

[12] B. Wydrowski, L. L. Andrew, and M. Zukerman, "Maxnet: A congestion control architecture for scalable networks," *IEEE Communications Letters*, vol. 7, no. 10, pp. 511–513, 2003.

[13] L. L. Andrew, K. Jacobsson, S. H. Low, M. Suchara, R. Witt, and B. P. Wydrowski, "Maxnet: Theory and implementation," *WAN-in-Lab project, pp1-11*, 2006.

[14] R. J. La, J. Walrand, and V. Anantharam, *Issues in TCP vegas*. Electronics Research Laboratory, College of Engineering, University of California, 1999.

[15] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[16] A. Baiocchi, A. P. Castellani, and F. Vacirca, "Yeah-tcp: yet another highspeed tcp," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42.

[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[18] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance." in *NSDI*, vol. 1, no. 2.3, 2015, p. 2.

[19] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[20] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[21] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.

Robustness of Scanner Exams with TUMexam

Simon Y. Kassahun, Stephan Günther*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: simon.kassahun@tum.de, guenther@tum.de

***Abstract*—Analysing the optimization possibilities for exams**

This paper reports on the opportunities to facilitate the conventional exam evaluation process by using scanner-based evaluation software such as TUMexam. It closely analyses the time cost and error probability for evaluating a regular exam to estimate the potential improvements of automated systems based on empirical data from a previously written exam.

***Index Terms*—TUMexam, exam evaluation**

1. Introduction

Exams are essential to most educational institution but are often very time consuming and messy. Many institutions rely on the conventional method of manually creating and evaluating exams. These require a lot of time from the faculty to be reviewed and also leaves students waiting for potentially weeks until they can find out their result. Manually calculating the test scores also carries the risk of making mistakes, which are not only annoying for the reviewers but can also potentially negatively impact a student's credit score. There is now a variety of commercial evaluation products available with many different approaches and scopes such as for example EvaExam[1] or eSystem[2]. With the intention to address the aforementioned problems, the Chair of Network Architectures and Services from the Department of Informatics at the Technical University of Munich began developing their own software called TUMexam[3]. TUMexam has been developed since 2015 with the aim to provide solutions ranging from templates and attendance records to facilitating the preparation for the evaluation and correction. Exams are currently still being manually corrected but each problem has boxes representing the amount of points awarded for a correct answer. Instead of writing down a number, the examiner ticks the corresponding checkboxes. After that step, all exams are scanned and then digitally analysed to count the collective score as well as calculate the final grade. The software is also being offered to other chairs. However, the decisive factor for most potential users is whether a switch to the new evaluation system brings a significant improvement to the processing time. This will be the main focus of this paper.

1. EvaExam - https://www.evasys.de/evaexam.html
2. eSystem - https://www.speedwellsoftware.com/exam-software/
3. TUMexam https://www.tumexam.de/

2. Methodology

To evaluate the robustness of scanner exams, it is compared to the conventional method, specifically to how often mistakes are made in a conventional exam evaluation and how much time could potentially be saved. One example for a conventional exam and one which uses TUMexam can be seen in Figure 1. The final exam from 2012 uses one box with two sectors for each (sub)problem. The two sectors are needed for the two correction passes. In contrast the final exam from 2017 which uses TUMexam has several boxes forming a table. The two columns are used for the two correction passes and each line represents 0.5 credits.

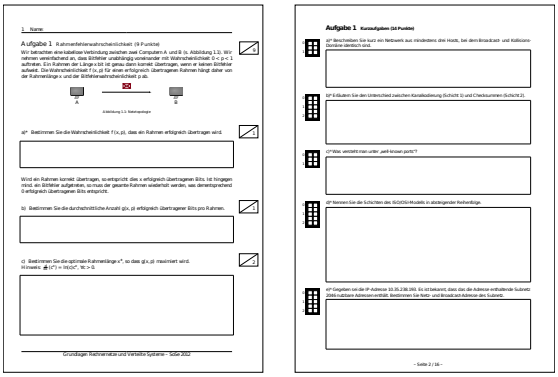

Figure 1. A sample page of the final exam GRNVS 2012 (left) and the the final exam GRNVS 2017 (right) [1]

Exams are normally corrected in two separated sessions. During the first correction pass the various subproblems are evaluated and given a score. In the conventional way, all these scores are manually summed up, for the most part by mental arithmetic, to give each problem a score. Finally all scores for each problem are summed up to calculate the total score but a calculator is used at this stage. The final score is listed on the cover along with the results of the individual problems which determine it. Since the final score is calculated from these summed up values, any previous error from one of the subtotals also affects the final score. During the second review, all subproblems are re-evaluated and the sums recalculated. In the case that all subproblems were given the same score, all sums should stay the same as well. If theses do not match, one of the two sums must be incorrect.

The significance of the aforementioned factors are assessed by repeating the counting process for a previous exam. That means counting all credits for each problem

doi: 10.2313/NET-2019-06-1_04

separately, once for the first correction pass and once for the second correction pass. The timer is started as soon as the correct page is opened to solely record the time spent on summing up credits. After each step the time is measured and it is noted whether an error occurred. During the first review the timer is stopped as soon as a result has been calculated. It is also recorded whether an error occurred or not but this is only done for evaluation purposes. The timespan for marking an error is not included in the noted time as it would normally be impossible to tell if the first correction pass was free of errors without a second correction pass. When such a deviation between the summed up credits and the score listed on the exam is found, it is noted which of these two numbers are incorrect and and by what margin. The second pass is very similar but the timer continues until it is clear which score is correct. Since a calculator is usually used to calculate the final score on the cover, a calculator is also used for that specific part.

There are two types of errors which are counted and evaluated in this paper. An exam error describes the case, where the score written on the original exam is incorrect. A counting error describes the case, where the calculated score is wrong.

TUMexam automates the process of counting scores and calculating the grade. Because exams are specifically designed for TUMexam, the software only needs to distinguish between a ticked box and an empty one. It also flags unclear marks for later review. Since no case is known so far where TUMexam calculated an incorrect score, number of counting errors is here assumed to be zero.

3. Implementation

The exam used for this test was a final exam with a time frame of 90 minutes. It is the final exam from the year 2011 of the course Introduction to Computer Networking and Distributed Systems. The exam has 5 different problems, each of which includes multiple subproblems making up the combined score for that problem. 192 individual exam sheets were reviewed for this paper.

3.1. Errors

Errors are to be expected and can be very troublesome for the correctors as well as the students. Since they can lead to much time being spent on finding the mistake up to potentially lowering a student's credits score when they go unnoticed, it is very desirable to keep the amount of errors and their impact as low as possible.

When calculating the credit score 2112 times (11 scores are calculated per exam), 81 individual errors were found in total. This group consists of 24 deviations in which the score written on the exam is false and 57 cases in which the deviation is a counting error. The difference between the wrong score and the actual score when an error occurred is very similar between the two type of errors. On average, errors on the exam differ from the correct score by 1.27 credits, while counting errors are off by 1.07 credits.

Also noteworthy is the overall distribution of the errors. As can be seen in Figure 2, almost no errors occurred when summing up credits in Section 4. This is explained by the fact that the fourth problem is the shortest one (see Figure 3), has the lowest attainable score, and consisted of only one double page.

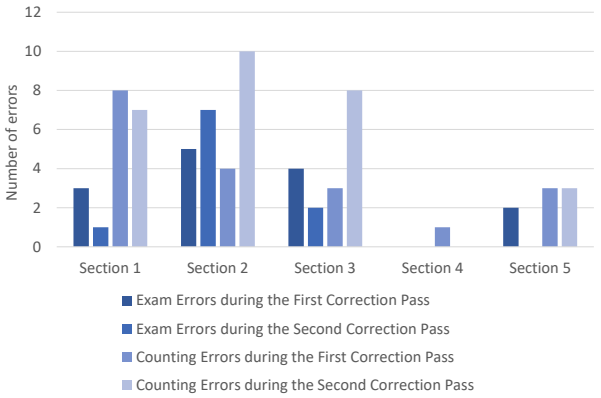

Figure 2. The number of errors per problem and type

3.1.1. Exam Errors. Out of the 24 cases where an exam had an incorrect score listed on it, 10 errors were found on the second correction pass. This effectively means that 0.86% of all problems have an incorrect score. In the worst case scenario, as many as 5.2% of the exams could have an incorrect credit score. The other 14 errors were made during the first correction pass.

3.1.2. Counting Errors. Counting errors did not occur consistently across the test. Only 19 errors were made during the first correction. Another 28 errors happened during the second correction and 10 when counting the combined scores on the cover.

3.2. Time

As the second key factor for an efficient and successful exam evaluation, a large time frame has probably a more noticeable impact and could potentially be greatly improved with automation. To find out how much time could be saved in the future, each part of a single exam sheet is measured individually and compared to the same part of the other correction pass and other exam sheets different. The results are shown in Figure 3. The dark blue graph represents the first pass and the light blue graph represents the second pass. The results are grouped together to improve visibility.

3.2.1. First correction pass. During the first correction pass the time needed to sum up credits solely relies on how long it takes the corrector to calculate a result. Since there is not any comparable credit score to verify the result, there is no need to recalculate the result again and rather helps identify errors for the second correctors when credit scores do not match. This influencing factor of needing to recalculate a score can also be observed in the small standard deviation of 5.1 s in relation to the entire first correction pass.

doi: 10.2313/NET-2019-06-1_04

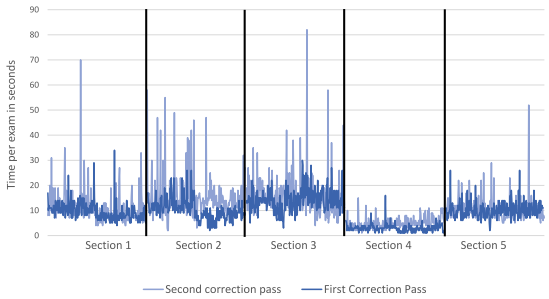

Figure 3. Comparison of the time spent on the two correction passes per section

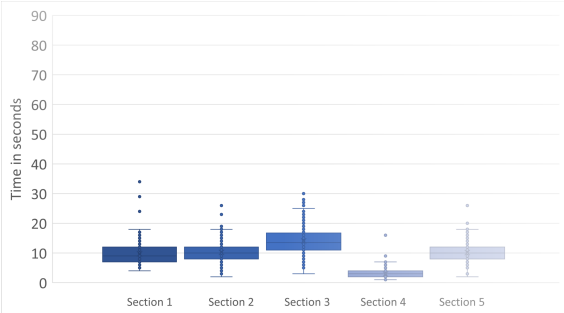

Figure 4. Box plot of the time spent per problem during the first correction pass

3.2.2. Second correction pass. The key difference of the second correction pass from the previous one it that errors made when summing up credits are noticeable to the corrector. The time spent on noticing such errors, finding the cause, and deciding which score is ultimately correct are now included in the measured time. When comparing these new results, we can see a significant increase of the variance. The median itself has not changed much.

Looking at the box plots (Figure 4 and 5), the individual problems have a very similar pattern but there are far more extreme outliers.

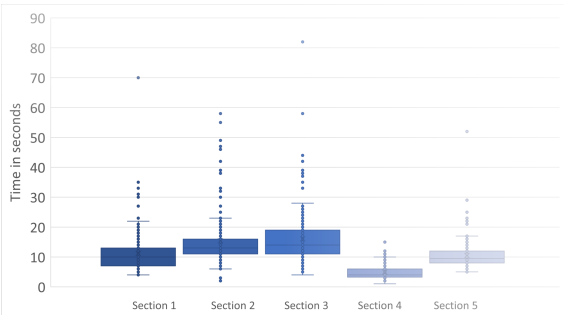

Figure 5. Box plot of the time spent per problem during the second correction pass

3.3. Estimations for an entire exam

The exam cover has two rows for both correction passes, however since they are distinct from each other, unlike the other parts of the exam, only the second correction pass was counted and then doubled when calculating the total time. Not taking any other factors, such as interruptions or navigating to the correct page, into account, a single exam takes on average 131.8 s. All 192 exams combined take 25 294 s, or approximately 7 h of pure counting time.

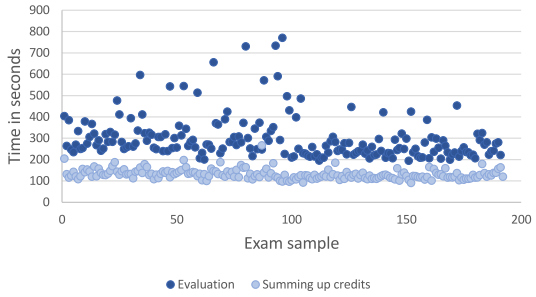

Figure 6. The time spent per exam

To put the calulated time for summing up all scores into perspective, it is compared to the time spent evaluating the same exam sheets. Since all records were saved along with their time stamps, it is possible to loosely reconstruct a more realistic time frame for procedures where exams need to be opened first and also takes other short interruptions into account. To make sure no major events affect this result, all breaks of longer than 10 min have been excluded. With these measurements an entire exam can take vaguely between 14 h and 15 h, slightly more than double the time it took to sum up credits. Figure 6 shows how the times compare for an invidiual sample.

3.4. Impact of errors on the time

To further analyse how big the impact of counting errors is, they are compared to the majority of samples where no error was made. Since errors do not have any affect on the first correction pass timewise, they are not included in this part. However, all errors which happened during the second correction pass are relevant but regardless of the type of error because both require more time. Both error types are therefore included. Figure 7 shows

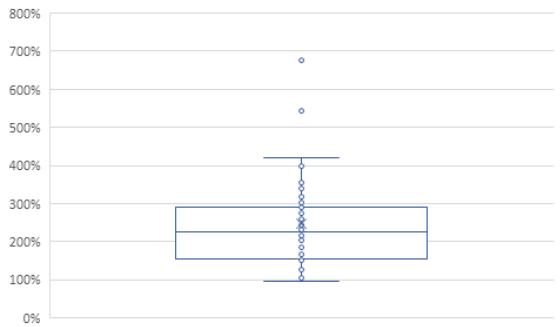

Figure 7. Box plot of the time spent per error relative to the average time spent on a sample of that problem where no error occurred

all cases where a errors was made and how much time they consumed relative to an average case of the same section when no errors was made. It shows that a case where an error occurs will on average take 246 % the time

doi: 10.2313/NET-2019-06-1_04

it would have taken if no error occured. The median is 224 %. Using the average time for a case without an error as a baseline, it is possible to calculate an estimate for the time all problems (without the cover) would require if the second correction pass was without any errors. The entire second pass consumed 3.1 h. Without and errors it would be around 2.9 h, saving approximately 12 min.

4. Conclusion and future work

An average 90 min exam with 192 participants and 5 problems takes under ideal circumstances require 7 hours to sum up all credits and calculate a final score. This number can be considered a rather low estimate and is very likely to increase a lot when including other factors. In a more realistic scenario you also have to factor in breaks, distraction and other interruptions. Depending on the workflow the time needed can increase to double the length or more. In this test only the time it takes to open the exam, note the test results and ordinary interruptions were factored in. Even though no breaks longer than 10 minutes were included, it raised the time required for the same exam evaluation significantly, to around 14 to 15 hours. While this is not an insignificant amount of time, it does also heavily depend on the number of participants as well as the scope of the exam.

0.86% of all problems had an incorrect score listed for the second correction pass. While this number might not seem significant, it is when put into the context of the entire exam. If each error occurred on a different exam, this would result in 5.2% of all exams having an incorrect score. None of the scores on the cover were calculated incorrectly (which does not include subsequent errors caused by incorrect scores from one of the problems). This is probably explained by the fact that a calculator is usually used for this part. One possible conclusion from this fact could be that using a calculator will likely decrease the number of errors. However using a calculator is also not risk free, as seen by the 10 counting errors which occurred when recounting the final score. Assuming all exam problems have a comparable credit score, a calculator also will not reduce the time by a significant margin as seen in Section 3.4

It is also worth mentioning that these evaluations are limited to the time consumed solely by summing up the credits. Streamlining other parts of the evaluation process, for instance by using software to automatically evaluate exercises and assist with the preparation of an exam, can also have huge advantages.

Conclusively, since TUMexam produces virtually no errors when summing up credits, there can be significant benefits to optimizing a conventional exam evaluation process in both time and error probability. It is likely that much smaller exams do not benefit from automation in a significant amount and even larger exams can expect to see much better results. However, this is out of the scope of this paper and could be evaluated in the future. Another possible direction could be to analyse other parts of the exam preparation and evaluation process, for example creating the exam problems, and how much time could then be saved by utilizing software.

References

[1] Archive of previous exams for the course "Introduction to Computer Networking and Distributed Systems at TUM", https://grnvs.net.in.tum.de/altklausuren/

doi: 10.2313/NET-2019-06-1_04

Network Resource Management for Virtual Networks with Learning Algorithms

Anton Mai, Advisor: Cora Perner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: anton.mai@tum.de, clperner@net.in.tum.de

***Abstract*—**
Network virtualization is a technique to have multiple virtual networks share resources of multiple substrate networks. This allows virtual networks to be decoupled from underlying hardware and leads to more flexibility. A big challenge of network virtualization is to efficiently manage network resources. There are various algorithms to allocate network resources to virtual networks. Recently, machine learning algorithms are proposed to manage network resources. This paper presents nonlearning algorithms and learning algorithms. Important aspects of both type of algorithm are discussed and compared. Many aspects show that learning algorithms can be more efficient in the long-term.

***Index Terms*—network virtualization, virtual network embedding, dynamic resource allocation, resource management, machine learning**

1. Introduction

The structure of the internet is dependant on the underlying physical infrastructure. Changing the structure is connected with high costs since hardware has to be replaced and added. Therefore the physical infrastructure is fixed. Network virtualization is a possible solution to the ossification of the internet. To increase the flexibility of the internet and networks in general, multiple virtual networks are mapped to substrate networks. The virtual networks share the resources of the substrate networks. Virtual network embedding is the process of efficiently embedding virtual networks to substrate networks with limited resources (Figure 1). There has been a lot of research on virtual network embedding. Embedding virtual networks is known to be NP-hard [21]. Another part of network virtualization is to dynamically allocate resources. Dynamic resource allocation is the process of allocating resources to different virtual networks during its runtime. Dynamically allocating resources requires the knowledge of multiple virtual networks to efficiently allocate resources without affecting the Quality of Service that the virtual networks provide. This leads to the need for monitoring the multiple virtual networks which requires more calculation and decreases the effiency of the whole system if the reward of the dynamic allocation is not high enough. Therefore past research had a higher focus on virtual network embedding than on dynamic resource allocation. However, recently there is research on dynamic resource allocation as it is becoming more interesting and efficient due to the use of machine learning. Machine learning became more popular in many different fields of computer science because of its nature of increasing its performance over time and its wide range of application cases, e.g. face recognition, language processing, finding the fastest path with consideration to the traffic. Machine learning is a technique based on the human ability to learn through experience. Machines simulate the learning ability of humans to learn by repeating certain actions and evaluating the reward of their actions. They gradually learn and improve over a long period of time until they are optimized. Recent works like [13] and [18] show many possible approaches to network virtualization by using learning algorithms to efficiently manage resources.

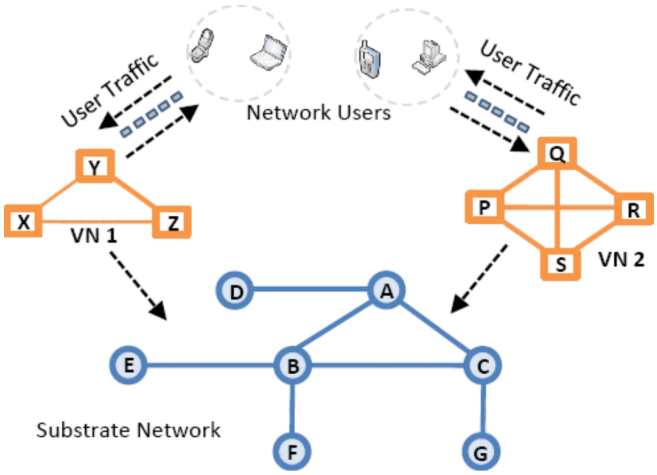

Figure 1: Virtual Network Resource Allocation, Figure 1 of [13]. The substrate network (hardware) consisting of nodes (A-G) and links is fixed. To be more flexible, virtual networks consisting of virtual nodes (P-S, X-Z) and links are mapped on the substrate network. Network users have access to the virtual networks without knowing about the underlying infrastructure of the substrate network.

This paper presents both nonlearning algorithms and learning algorithms for network virtualization and shows different aspects of both kind of algorithms. The high potential of learning algorithms, what possibilities they present and how their limitations can be remedied are shown in this paper. The rest of the paper is structured as follows. In section II different approaches to nonlearning algorithms and their characteristics are presented, in section III recent learning algorithms are presented. In section IV the aspects of nonlearning and learning algorithms are compared. Section V discusses the results of section V and summarizes the results of this paper.

doi: 10.2313/NET-2019-06-1_05

2. Nonlearning Algorithms

Virtual network embedding has been a well known problem. There are various approaches to the virtual network embedding problem.

Policy-based: Miyamura, Kamamura and Shiomoto [1] propose a policy-based approach to resource management with each virtual network having its own reserved resources. If needed, a virtual network could get access to resources that are shared between them.

Divide and conquer: Zhang et al. [3] propose an approach to divide substrate networks into many partitions to deal with large-scale networks.

Distributed multi-agent architecture: Soares and Madeira [4] developed a dynamic, distributed multi-agent architecture where each agent is located in the substrate network nodes. This leads to less complexity as actions are performed locally and automatic actions by agents in the form of self-management and self-healing.

Cognitive: Han et al. [5] propose a cognitive management scheme for managing virtual network resources that focuses on the topology and centrality.

Hierarchical Katayama et al. [6] use a hierarchical approach to the virtual network embedding problem. Submanagers are used, which manage multiple substrate network nodes to reduce complexity.

Sharing-based: Mao et al. [8] propose a sharing based network embedding algorithm where network resources are divided into equal time slots before starting with the embedding process.

Prediction-based

The nonlearning approaches are often based on assumptions that the demand of resources from the virtual networks do not change much. Therefore it is rather unflexible, limiting these algorithms to having to know the amount of allocated resources beforehand. As stated by Mijjumbi et al. [13] most of the approaches are static and focus on embedding virtual networks. The allocated resources are not changing during the lifetime of the virtual network. There is no dynamic allocation of resources, in some cases there is a possible migration of the virtual links between the virtual nodes, but the allocated resources are fixed. Reconfiguring the mapping of virtual networks to substrate networks is not available or only in case of failures. But if the demand of resources is not changing much during its lifetime, nonlearning approaches are fitting for the virtual network.

3. Learning Algorithms

Recently there have been many approaches to virtual network resource management that use machine learning to allocate network resources from substrate networks to virtual networks. Most of the learning algorithms are based on reinforcement learning and neural networks. Reinforcement learning is a method to make an agent learn based only on rewards it gets for its actions. Based on past actions, a utility function is approximated, showing the utility of each action in each state. Q-Learning is a reinforcement learning technique.

Q-Learning: Mijumbi et al. [13] proposed an approach to use Q-learning to dynamically manage resources after embedding the virtual network. An agent chooses an action based on the Q-values Q(s,a) of each action in that state. The action is chosen random, but actions with a higher Q-value have a higher probability to be chosen. After each learning episode, the Q-values are updated based on the received reward. The Q-values are updated based on the Q-learning rule in (1).

$$Q(s_p, a_p) \leftarrow (1-\alpha) * Q(s_p, a_p) + \alpha \left\{ r_p + \lambda max Q(s_n, a) \atop a \in A \right\} \tag{1}$$

The parameter s_p represents the present state, a_p the present action, s_n the next state and r_p the immediate reward for taking the action a_p. The updated Q-values are comprised of the past Q-value and the new reward. The parameter $0 \leq \alpha \leq 1$ determines how fast the agent is learning. Having a learning rate of α closer to 1 makes the agent learn more from new experiences but also past experiences get less important. Therefore α is often set near 0 to make the agent learn slowly but steadily. The discount factor λ determines whether immediate rewards or future rewards are more important. A discount factor of λ closer to 1 gives more important to future rewards. With each learning episode, the Q-values converge towards an optimal solution.

Mijumbi et al. used 8 different values to describe the percentage of used resources. Each state is represented by the percentage resource allocation, the percentage of unused virtual resources and the percentage of unused substrate resources. So there are 8 * 8 * 8 = 512 different states. They used 9 different actions to change the percentage of used resources, so there are 9 * 512 = 4608 different state-action pairs and Q-values.

Autonomic and distributed: Mijumbi et al. [11] propose an autonomic and distributed way to manage resources in virtual networks. Each virtual network is managed by an autonomous agent. The agents use reinforcement learning and they cooperate with each other to automatically manage the resources. These virtual networks can heal, configure, protect and optimize themselves through reinforcement learning. **Ant colony optimization:** Cao et al. [7] propose an ant colony optimization algorithm to optimize virtual network embedding. The ant colony optimization algorithm mimics ants that follow pheromones secreted by other ants to get to their food. Paths that are more frequently used contain more pheromones, so by following paths with stronger scents of pheromones, the ants are improving their effiency until it is optimal.

Neuro-fuzzy: Mijumbi et al. [12] propose a neuro-fuzzy approach to manage network resources. It is an reinforcement based approach that is distributed and dynamically allocates resources to the virtual networks.

Autonomous neural network Mijumbi et al. [14] also proposed an autonomous neural network based resource allocation management.

Statistical Learning: Li [15] developed a dynamic resource management approach based on statistical learning that guarantees no violation of the Quality of Service of the virtual networks.

Radial basis function neural network: Zhang et al. [18] use a radial basis function neural network to embed the virtual networks. By using training samples they simulate

doi: 10.2313/NET-2019-06-1_05

an embedded virtual network and learn with the simulation. After the learning period the virtual network is embedded based on the expected usage of resources.

Learning algorithms are mainly used to dynamically allocate resources throughout the runtime of the virtual network and to improve the network. Also as seen in case of [18], radial basis function neural networks can be used to simulate the training process, presenting the possibility to apply learning algorithm on the embedding of virtual networks. Learning algorithms have high potential as they approach nearly optimal solutions after enough learning time. Learning algorithm are probabilistic so they can try other steps and possibly learn through these experiences. That leads to the problem that learning algorithm can never be optimized as there is always some probability that it does not follow the optimal strategy. Another big problem of learning algorithms is their initialization. At the time of initialization the virtual networks have no knowledge as they had no time to learn. The period of time or number of test samples that is needed to learn an sufficient efficient strategy can be quite big. So at the beginning learning algorithms are bound to fail a lot. Having a virtual networks that is expected to fail at the start could become a problem.

4. Comparison of nonlearning and learning algorithms

Most nonlearning algorithms are focused on the embedding of the virtual network. There is less focus on dynamically reallocating virtual network resources during the runtime of the virtual network while learning algorithms are mainly used to dynamically allocate resources during the runtime. Nonlearning algorithms are based on assumptions. If the assumptions are right and do not change significantly during the runtime of the network, then nonlearning algorithms are efficient. Learning algorithms are more dynamic and flexible in comparison to nonlearning algorithms, so for changing demands of resources they are more fit. Also being able to predict changing demand of resources and proactively adjusting to changes is a big advantage of using learning algorithms. As shown by Mijumbi et al. in [13] the number of accepted networks is much larger with dynamic learning algorithms than with static nonlearning algorithms (Figure 2). The biggest disadvantage of learning algorithms in comparison to nonlearning algorithms is their bad performance at the initialization state and the large period of time needed to become as efficient as nonlearning algorithms. Also shown in [13], for the packet drop rate, dynamic learning algorithms need some time to learn to be as efficient as static approaches (Figure 3). Another minor disadvantage of learning algorithm is that while learning there is some minor work put in learning, in comparison to nonlearning algorithms which do not work proactively. Mijumbi et al. in [14] compared their artificial neural network with a dynamic approach based on reinforcement learning and two static approaches (Figure 4). Simulations showed that artificial neural networks are significantly better that the other approaches. Also, the dynamic approach based on reinforcement learning showed better results than both static approaches.

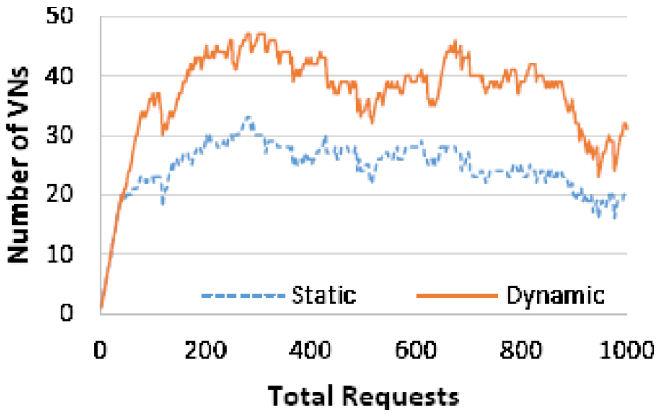

Figure 2: Number of Accepted Networks, Figure 9 of [13]

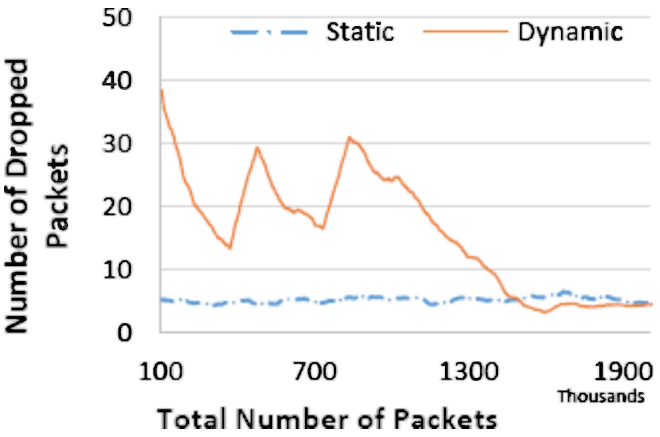

Figure 3: Node Packet Drop Rate Variation, Figure 10 of [13]

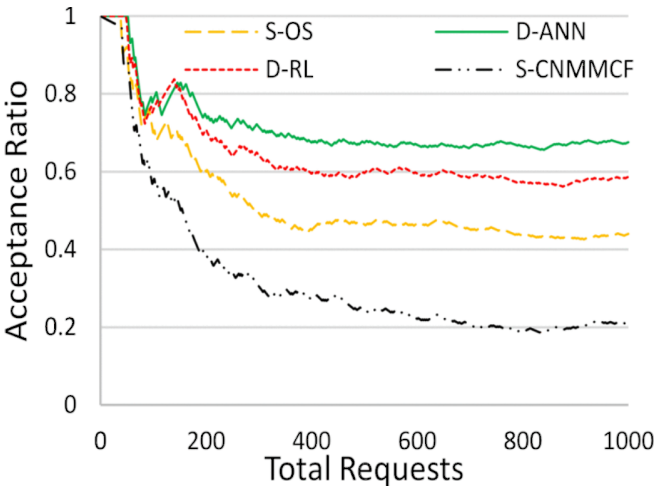

Figure 4: Acceptance ratio, Figure 3 of [14]. The two dynamic approaches D-ANN (Dynamic, based on Artificial Neural Networks) and D-RL (Dynamic, based on Reinforcement Learning) were compared to two static approaches S-CNMMCF (Static, Coordinated Node Mapping and MCF for link mapping) and S-OS (Static, link based optimal one shot Virtual Network Embedding). It is visible that the dynamic approaches perform much better than the static ones.

doi: 10.2313/NET-2019-06-1_05

5. Conclusion and Future Work

In this paper various approaches to managing resources in network virtualization were presented. First traditional nonlearning algorithms to solve virtual network embedding were explained, then recent research on the topic learning algorithms was presented. Both types of algorithms showed potential in managing network resources, but in comparison learning algorithms are more useful in most cases and promise more potential in effiency.
Nonlearning algorithms have a short-term advantage over learning algorithm because of the weak performance of learning algorithms at the initialization state. Also in networks without significant changes in network resource demand, nonlearning algorithms perform well from the start. However for most networks, especially for the future internet the ability to dynamically reallocate resources will be essential. Learning algorithms show great potential to improve efficiency and flexibility. Mijumbi et al. adress in [16] the disadvantage of learning algorithms at the start and suggest following solution: Initiating an offline learning step to let the learning algorithm improve first. This solution would remedy the bad performance of learning algorithm at the start. This solution is similar to the approach of Zhang et al. in [18], to use other virtual networks as test samples to nurture their radial basis function network algorithm in a first step before embedding the virtual networks. Therefore learning algorithms would be advantegeous to nonlearning algorithms in most cases, if a first learning step is introduced. For future work it would be interesting to research if there are different patterns of agents using learning algorithms. This could shorten the time needed to learn as different evaluations for each action can be implemented from the start. Also, it would be interesting to research how to optimize using learning algorithms on networks with human users. Maybe learning algorithms could also be used to predict the usage of the network of different people based on their past usage.

References

[1] T. Miyamura, S. Kamamura, K. Shiomoto, “Policy-based resource management in virtual network environment”, in 2010 International Conference on Network and Service Management (CNSM)

[2] S. Xiaochuan, C. Hongyan, C. Jianya, L. Yunjie, “IDP-VRMA: An intelligent and distributed virtual resource management architecture based on prediction for future networks”, in 2011 International Conference on Advanced Intelligence and Awareness Internet (AIAI)

[3] S. Zhang, X. Qiu, L. Meng, “Virtual network mapping algorithm for large-scale network environment”, in 2011 6th International ICST Conference on Communications and Networking in China (CHINACOM)

[4] M. A. Soares, E.R.M. Madeira, “A multi-agent architecture for autonomic management of virtual networks”, in 2011 IEEE Network Operations and Management Symposium

[5] Y. Han, Z. Wang, H. Tang, S. Ci, “A novel cognitive management scheme for the virtual network resources”, in 2012 IEEE Globecom Workshops

[6] Y. Katayama, K. Yamada, K. Shimano, A. Nakao, “Hierarchical resource management system on network virtualization platform for reduction of virtual network embedding calculation”, in 2012 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)

[7] Cao W., Wang H., Liu L. (2014) An Ant Colony Optimization Algorithm for Virtual Network Embedding. In: Sun X. et al. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2014. Lecture Notes in Computer Science, vol 8630. Springer, Cham

[8] Y. Mao, Y. Guo, H. Hu, Z. Wang, T. Ma, “Sharing Based Virtual Network Embedding Algorithm With Dynamic Resource Block Generation”, in 2015 IEEE Communication Letters (Volume 19, Issue 12)

[9] J. Li, Y. Wang, Z. Wu, S. Feng, X. Qiu, “A prediction-based dynamic resource management approach for network virtualization”, in 2017 13th International Conference on Network Service Management (CNSM)

[10] M. Lu, Y. Lian, Y. Cheng, M. Li, “Collaborative Dynamic Virtual Network Embedding Algorithm Based on Resource Importance Measures”, in 2018 IEEE Access

[11] R. Mijumbi, J. Serrat, J. Gorricho, “Autonomic Resource Management in Virtual Networks”, in arXiv:1503.04576 [cs.NI]

[12] R. Mijumbi, J. Gorricho, J. Serrat, M. Shen, K. Xu, K. Yang, “A neuro-fuzzy approach to self-management of virtual network resources”, in Expert Systems with Applications (Volume 42, Issue 3)

[13] R. Mijjumbi, J. Gorricho, J. Serrat, M. Claeys, F. Turck, S. Latré, “Design and evaluation of learning algorithms for dynamic resource management in virtual networks”, in 2014 IEEE Network Operations and Management Symposium (NOMS)

[14] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, J. Famaey, F. Turck, “Neural network-based autonomous allocation of resources in virtual networks”, in 2014 European Conference on Networks and Communications (EuCNC)

[15] Y. K. Li, “QoS-Aware Dynamic Virtual Resource Management in the Cloud“, in Applied Mechanics and Materials, Vols. 556-562, pp. 5809-5812, 2014

[16] R. Mijumbi, J. Serrat, J. Gorricho, “Self-managed resources in network virtualisation environments”, in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)

[17] A. Blenk, P. Kalmbach, P. Smagt, W. Kellerer, “Boost online virtual network embedding: Using neural networks for admission control”, in 2016 12th International Conference on Network and Service Management (CNSM)

[18] H. Zhang, X. Zheng, J. Tian, Q. Xue, “A Virtual Network Embedding Algorithm Based on RBF Neural Network”, in 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)

[19] T. Miyazawa, V. P. Kafle, H. Harai, “Reinforcement learning based dynamic resource migration for virtual networks”, in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)

[20] He, M., Zhuang, L., Tian, S. et al. J Wireless Com Network (2018) 2018: 150. https://doi.org/10.1186/s13638-018-1170-x

[21] A. Fischer, J. Botero, M. Beck, H. Meer, X. Hesselbach, “Virtual Network Embedding: A Survey”, 2013 IEEE Communications Surveys & Tutorials

From FIFO to Predictive Cache Replacement

Daniel Meint, Stefan Liebald*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: d.meint@tum.de, liebald@net.in.tum.de

***Abstract*—Caching is an important technique to accelerate data reads in various hardware and software systems. The choice of a replacement policy to decide which item to evict in order to make space for newly requested data is at the core of every cache design. A vast number of heuristics have been proposed in the literature. This paper gives an overview of some of the most popular replacement mechanisms. The strategies are classified and described. An exhaustive taxonomy of traditional strategies is proposed and explained. The paper also presents the Adaptive Replacement Cache and a predictive cache replacement strategy that was designed specifically with multimedia Web traffic characteristics in mind. Further, techniques that are closely related to the cache replacement issue, especially in Web caching, are discussed.**

***Index Terms*—cache replacement strategies, network caching, adaptive replacement cache, predictive cache replacement, multimedia content delivery**

1. Introduction

Caches are employed to *localize traffic* by temporarily storing data close to the consumer and, today, they are ubiquitous in multiple areas of computing. Hardware-managed caches are used by the CPU and the GPU, for example. Software caching is found in database systems and on the Web. Generally, cached information can be retrieved significantly faster than if the origin storage would have to be consulted. However, cache capacity is typically limited and only a fraction of the existing resources can be stored at any point in time.

When data is requested, the *cache client*, such as a Web browser or a CPU, checks its closest cache first. If the information is available in the cache, we refer to this as a *cache hit*. If not, a *cache miss* occurs and data has to be read from lower-level memory. This lower-level memory can either be the original location of the resource, like a Web server, or, in case of a hierarchy of caches, a different, lower-tier cache.

When missing information needs to be brought into the cache and the cache is already full, old objects must be removed. The "victim" can be drawn randomly or chosen in a deterministic process. The respective heuristic is called the *cache replacement strategy* (also *cache replacement policy*, *eviction policy* or *removal policy* [6]) and is a central component of every caching scheme.

This paper provides an overview of extensively researched cache replacement strategies and further describes limitations and challenges of caching large objects, like multimedia content. The focus is hereby drawn towards *Web caching*.

The next section describes how cache replacement strategies can be evaluated and compared. Subsequently, some simple cache replacement approaches that partly originated from traditional disciplines of computing are discussed. Section 4 then presents the Adaptive Replacement Cache, a more recent proposal to cope with dynamic access patterns. Finally, Section 5 addresses the challenges of caching multimedia content on the Web and outlines how prediction-based replacement can be particularly effective to tackle these challenges.

2. Evaluation of Cache Replacement Policies

Mathematical models exist to evaluate the performance of cache replacement algorithms [1], [11]. Competitive Analysis compares the performance of an algorithm with the best possible performance [24]. The resulting *competitive ratio* corresponds to "the maximum ratio of the algorithms cost to the optimal offline algorithm's cost over all possible request sequences" [1]. An *offline* algorithm knows the entire request stream from the very start and can always make the best possible decision. It, therefore, only serves as a theoretical upper bound on the achievable performance by any *online* policy. Competitive Analysis merely studies worst-case scenarios and is difficult, especially when documents can be of variable size [8].

It is more common to conduct experimental studies to compare different replacement strategies. An established method is to run a *trace-driven simulation*. Performance is hereby studied on realistic cache traces, which often gives more practical insights than theoretical upper and lower bounds [8]. The following metrics are amongst the most commonly reported measures:

- **Hit Ratio (HR):** The fraction of all client-requested objects that could be served from cache, e.g. a hit rate of 10% means that one out of every ten requests resulted in a cache hit.
- **Byte Hit Ratio (BHR):** The fraction of all client-requested bytes that could be served from cache. Sometimes this value is also referred to as a "weighted" hit ratio [6]. Because it takes object size into account, BHR indicates bandwidth savings better than HR.
- **Delay Savings Ratio (DSR):** DSR reports the reduction of client-perceived latency. Its calculation

doi: 10.2313/NET-2019-06-1_06

is subject to external factors, like network congestion and server stability. Precise measurement of download delays is difficult and results often fluctuate [7].

The performance of policies in trace-driven simulations is sensitive to the character of the employed trace. Wong [11] notes that inconsistent, and even contradictory results have been reported in the literature.

3. Traditional Cache Replacement

This section describes some conventional approaches to cache replacement. First, we name factors that may influence the replacement decision. Then we present some example strategies. For the purpose of this paper, traditional strategies are divided into the following groups:

- **Key-Based Strategies** sort objects upon a *primary key*. Ties are broken using secondary, tertiary or even more keys.
- **Function-Based Strategies** incorporate multiple weighted factors, in no sequential order but concurrently in a specific function that calculates the *value* of every object. The weighting parameters may be fixed or dynamically adapting to the properties of the access stream [8].
- **Randomized Strategies** use nondeterministic algorithms to remove entries. Randomized policies typically perform worst in most scenarios [11] but also require the least resources. They are therefore primarily used in systems with severely limited processing power. Since randomized policies do not keep meaningful state information, we consider this brief introduction to be sufficient and will not discuss them any further.

Among the key-based policies, we further distinguish strategies according to which keys they actually consider. This taxonomy roughly follows and combines the proposals made by Wang in [2], Podlipnig and Boszormenyi in [7], and Balamash and Krunz in [8]. Various other classifications have been used in related work, for example in [9], [38], [54].

3.1. Influencing Factors

The following keys are universally used in cache replacement to characterize cached items and determine their utility [7]:

- **Arrival:** When was an object admitted to the cache? Typically, new items are favored to stay in the cache.
- **Recency:** When was the last request for an object? Recently accessed items are favored.
- **Frequency:** How often has an object been requested? Frequently accessed items are favored.

Network traffic has certain characteristics that suggest incorporating additional keys for replacement decisions in Web caching. CPU and disk caches are typically concerned with the management of uniformly sized blocks known as *pages*. In contrast, resources from the Web are stored as whole Web *documents* and can vary greatly in size [1], [35]. Certain items may, therefore, take up a disproportional percentage of the cache's total capacity, which should be considered by the replacement mechanism. Further, the effort of obtaining information over a network is not only correlated to the data volume, but other dependencies, such as bandwidth and distance, make the calculation of a *cost* factor more complex. Finally, cached information can also become outdated. Especially HTML resources of popular websites are updated relatively frequently [8]. The following additional keys can help Web cache replacement algorithms make more informed decisions:

- **Size:** How much space does an object occupy? Small files are favored.
- **Cost:** How expensive would it be to re-fetch an object? Possible metrics include hop count and bandwidth along the delivery path, expected latency, and monetary cost. Expensive items are favored.
- **Expiration:** When is an object presumably going to become stale? Items with a long validity are favored.

3.2. Key-based Policies

Key-based policies sort their candidates upon a primary key. If the primary key does not guarantee to determine a single clear winner, i.e., multiple objects can have identical values, a secondary key is necessary to break ties. If objects could tie on the second factor as well, a tertiary key is consulted, and so on.

First In, First Out (FIFO) is the simplest arrival-based strategy. Objects leave a queue in the order in which they arrive. Hence, this basic approach always evicts the oldest object from the cache. FIFO can easily be implemented with constant computational and optimal space overhead, and is, therefore, suitable for systems with strictly limited computational power or storage capacity. FIFO completely ignores both recency and frequency of access when making decisions. Generally, in practical applications, FIFO is significantly outperformed by policies that take these factors into account [17], [18].

The SIZE [6] strategy evicts the largest objects first with the intent to make space for multiple smaller files. Favoring small files results in a higher number of total files cached and, thus, a good file hit ratio but decreased byte hit ratio [11], [35], [53]. As mentioned earlier, BHR is closely related to bandwidth savings, so if the cache's goal is to minimize download volume from the Internet, for example, this behavior is undesirable. With a priority queue based on object size, eviction can be performed in logarithmic time.

Because the subsequent discussion of ARC in Section 4 builds on a thorough understanding of recency- and frequency-based policies, we discuss these concepts in the following dedicated subsections.

3.2.1. Recency-based Policies. Recency-based policies sort objects according to how recently they were requested. The underlying rationale is that recently accessed information is more likely to be demanded again in the near future and should, therefore, be retained in the cache.

Information that has not been useful for the longest time is accordingly regarded as least valuable and should be evicted first.

For this rationale to hold, access streams need to exhibit *temporal locality*, i.e., the past and the future must not be independent, but resources become "hot" (accessed frequently) and cool down again [7], [20]. Jin and Bestavros find that temporal correlations are present in Web traffic and exist most dominantly in the *short-term* [37], [38]. Recency-based strategies are therefore most effective in caches of small sizes and high activity [11].

Caches may form a multi-level hierarchy. Rajan and Ramaswamy describe how temporal locality is inherent to first-level caches but decreases throughout a storage hierarchy in [26]. The most recently accessed files are assumed to be served from the first-level cache anyway and so, for lower-tier caches further down the delivery path, e.g. in root-level proxies, locality features are already "filtered out". Here, other heuristics can perform better than recency. Busari and Williamson demonstrate that size-sensitive policies are more effective in root-level caches in [53].

The *Least Recently Used* (LRU) strategy can be seen as the originator of recency-based replacement. It simply evicts the object that was not accessed for the longest time. Assuming requests are processed sequentially, every request has a unique timestamp. Hence, we do not need additional tie-breaker keys. LRU can be implemented with a linked-list data structure supported by a hashing mechanism for lookup. Its simplicity and constant time complexity make LRU particularly attractive for hardware caches.

LRU's biggest threats are large sequential reads of data that is only needed once and then never accessed again. Since LRU only takes recency into account, it degenerates to FIFO and these "scan" sequences quickly flush out potentially more valuable items and pollute the cache.

Many variants of LRU have been proposed. One strategy proposed by Pitkow and Recker [25] uses a different time granularity. The authors observe that client interests change on a daily basis. As a consequence, they suggest using the number of full days since the last request as a primary factor. If there are no objects older than one day, size serves as a secondary key. The largest files are replaced first.

LRU-Threshold [27] rejects files larger than a specified threshold before they can even get into the cache. It could, therefore, be argued that the primary key is size. When cached files need to be removed, the victim is determined according to LRU. Hence, we consider the actual replacement process to be recency-based. This classification agrees with the suggestions made in [11].

Other recency-based strategies include LRU* [28], LRU-Hot [29], Segmented LRU (SLRU) [30] and HLRU [32].

3.2.2. Frequency-based Policies. Frequency-based policies sort objects according to how often they have been requested in the past. The underlying rationale is that some data is consistently more popular than other data, i.e., exhibits *long-term* popularity, and information that was frequently requested in the past will keep being accessed a lot in the future. Tiebreaker policies are unavoidable because multiple objects can easily have identical frequency values.

Among Web documents, popularity distribution follows Zipf's law [4], [33]. This means that only a small set of very popular items accounts for a significant fraction of the overall traffic. Keeping the "hottest" items in the cache should be sufficient to satisfy most requests. However, when the set of popular data changes abruptly, frequency-based strategies cannot adapt as quickly as recency-based strategies [11]. Frequency-based heuristics perform best in environments with static popularity characteristics, i.e., popular data stays popular and unpopular data stays unpopular [7].

Frequency-based strategies are normally implemented with a priority queue [7] offering logarithmic time complexity per operation.

The *Least Frequently Used* (LFU) strategy always evicts the item with the lowest frequency count. Podlipnig and Boszormenyi distinguish between two forms in [7].

- **Perfect LFU** keeps track of all requests to objects ever recorded. This form is of theoretical value, but unbound space overhead makes its implementation infeasible.
- **In-Cache LFU** keeps track of requests to currently cached objects only. Once an item is removed, the count is forgotten. This form has imperfect information, but the space overhead stays manageable.

Unlike LRU, LFU is *scan-resistant*. Frequency is of relevance, so sequentially accessed items only replace each other.

However, especially In-Cache LFU suffers from a different cache pollution problem. Objects that were popular at one point in the past, and have accumulated high reference counts, hardly get flushed out, even if they are currently unpopular. Newer, potentially more useful (hotter) items have a hard time to stay in the cache because they start off with the lowest possible score. Again, this suggests that LFU performs best in systems where the demand for an object stays at a constant level.

More versatile alternatives avoid the cache pollution problem by also incorporating recency into the decision-making process. So-called *aging* mechanisms effectively decrease an object's value when it has not been requested for a certain amount of time. Recently accessed items are consequently regarded as more valuable and remain in the cache. *LFU with Dynamic Aging* (LFU-DA) [31] is an example of a hybrid strategy that combines frequency and recency aspects.

3.3. Function-based Policies

Function-based removal policies consider multiple attributes at once and not separately. So there are no more primary and secondary keys but instead a single, usually nonnegative value H associated with each cached object i.

One popular member of this category is the *GreedyDual-Size* (GD-Size or GDS) algorithm [1]. When an item enters the cache, its value is initialized as follows.

$$H_i = cost_i / size_i$$

doi: 10.2313/NET-2019-06-1_06

When the cache has filled up, the item with the lowest value $H_{min} = min_{j \in cache} H_j$ is removed. Subsequently, the values of all remaining items get reduced by H_{min}:

$$\forall i : H_i \leftarrow H_i - H_{min}$$

So their scores shrink over time. On a cache hit, the value of the re-requested item gets reset to its original one.

The $cost$ parameter can be defined in various ways, depending on the objective of the caching system [8], [34]. For example, setting $cost_i = 1$ uniformly for all objects means an item's initial H value corresponds to $1/size_i$. Due to this direct inverse correlation of value and size, the largest files are regarded as least sustainable and get removed in favor of multiple smaller files. Consequently, this definition of cost maximizes the total number of files cached and, therefore, results in the best hit rate [35].

Depending on the environment, other cost definitions are possible. For network caching, GD-Size might instead define cost as the number of packets involved in a retrieval, the expected latency increase or the number of hops between the cache and the origin server [10].

The GreedyDual-Size policy is an extension of the original GreedyDual algorithm introduced by Young [36]. Today, an entire range of algorithms, referred to as the "GreedyDual-family" [10] exists. Members include GD-Frequency, GD-Size-Frequency [31], and GD*, an adaptive generalization of GD-Size [38].

Some function-based policies allow parameters to adapt to workload characteristics [7]. This means that when requests are observed to have high temporal correlations, for example, recency should be weight heavier than other factors. If the access stream indicates that popularity levels are rather steady, on the other hand, the underlying function should put more emphasis on frequency counts. The downside of this adaptive functionality is increased computational complexity [7], which could hurt performance overall, especially when adaptiveness is unnecessary.

4. Adaptive Cache Replacement

This section presents the *Adaptive Replacement Cache* (ARC) [12]. ARC neither associates H values with entries like function-based approaches nor should it be categorized as key-based in the traditional sense as presented above, since it does not apply a fixed sequence of primary key, secondary key, etc. Instead, it constantly reacts to changes in the character of the processed requests to balance recency and frequency aspects in a *self-tuning* manner, i.e., there are no parameters that need to be set manually.

ARC cleverly combines two LRU-lists of varying size and a history of recently evicted items to make removal decisions.

When an item enters the cache, it is placed at the *most recently used* (MRU) position of the recency-ordered list L_1. L_1 is therefore considered to contain recently required items. If the item gets requested a second time, while cached, it is considered to be frequently accessed, and "promoted" to a second list L_2, again, entering at the MRU position.

Both lists are further split into a "top" and "bottom" part, that is, L_1 consists of two sublists T_1 and B_1 and L_2 is the union of T_2 and B_2.

The top sections form the *main cache* and store full objects that can be returned as expected.

The bottom sections form the *ghost cache* and only store identifiers, i.e., metadata, for the objects that once were in the main cache, but got flushed out. The ghost cache merely serves a history function. It cannot provide the resource data to answer client requests.

Now, the lists T_1 containing items that have been accessed once recently and T_2 containing items that were requested at least twice compete for cache capacity. Depending on the workload attributes they grow and shrink constantly. As the full-fledged balancing process is fairly complex, we will only explain the basic idea underlying ARC's adaptability and refer to [12] for a detailed description. Intuitively, when a request reaches the cache, the following events influence ARC's behavior.

If the requested item is not contained in the main cache, but a "ghost hit" in B_1 occurs, ARC concludes that recency features are currently important. The request cannot be satisfied from cache directly because the item was pushed out of T_1 into the ghost section, only retaining metadata. If T_1 was granted more capacity, the request might have resulted in a "real" cache hit. Therefore, T_1 grows and T_2 shrinks, i.e., the least recently used item in L_2 will be evicted next.

If the request hits the B_2 ghost cache, ARC considers frequency aspects to currently be neglected. Ghost hits in B_2 therefore let T_2 grow at the expense of T_1, i.e., the least recently used item in L_1 will be evicted next.

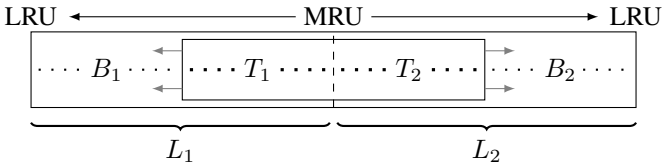

Figure 1. Simplified ARC cache directory visualized in a balanced state. The inner rectangle represents the actual cache, which is fixed in size but can freely move across the history sections. Items enter the cache at the center and get gradually pushed outwards unless they are re-accessed.

ARC is considered scan-resistant since the separation of items that have been accessed only once ($L1$) and items that have been accessed at least twice recently ($L2$) protects the latter section of being flooded by single-access streams.

It was reported to reliably and substantially outperform established mechanisms like LRU in trace-driven simulations [12], [13]. ARC does not require significantly more space than LRU and has the same, constant time complexity.

Since 2006, IBM holds a patent for the Adaptive Replacement Cache [15], which has complicated its deployment for third parties [16].

5. Predictive Cache Replacement for Web-Based Video Content

The above-described algorithms were mostly designed as general-purpose policies. LRU and GDS have been

doi: 10.2313/NET-2019-06-1_06

titled "good enough" or "champion" algorithms [1], [34] because they perform sufficiently well on the basic performance metrics for caches of various sizes and environments. Wong states that cache replacement in its general form is a "solved topic" [11] and further research would only lead to marginal improvements.

However, more recently, authors have also depicted that "the network environment is dynamic and uncertain" [34] and future requirements may very well overwhelm the identified "good enough" algorithms. Podlipnig and Boszormenyi say, their sufficiency for managing multimedia content is "questionable" [7].

This section presents a more specialized caching approach to cope with the challenges of serving multimedia data on the World Wide Web. Video streaming is arguably the most extreme cases. It combines audio and imagery, requiring storage volumes and retrieval bandwidths multiple magnitudes greater than ordinary text and small graphics. The focus of this section is, therefore, on Video-on-Demand (VoD) services. First, the unique properties of video data and its consumption are described. The second subsection then presents a novel cache replacement proposal by Famaey et al. [51] that predicts the future popularity of multimedia content.

5.1. Multimedia Caching

From a storage perspective, multimedia files differ from text-based data most notably in volume. Cached objects are significantly larger in size, as compared to when cached information is text-based [11], [39]. Thus, we expect space for storage and bandwidth for transport to be the critical factors for multimedia applications. Further, caching mechanisms need to guarantee continuous delivery of video without stutters for enjoyable consumption. Dan and Sitaram conclude that traditional replacement is insufficient for video caching requirements [39].

On the other hand, multimedia data offers the possibility of significantly reducing volume "without sacrificing too much quality" [7]. Text files typically need to be compressed lossless to allow perfect reconstruction. In video compression, inter-frame techniques eliminate redundancy without compromising quality [41]. Even noticeable quality losses may sometimes be acceptable when this results in a smoother presentation.

Converting files from one representation to another, e.g. format conversion or compression, is called transcoding [42]. On the Web, *transcoding proxy servers* [42], [45] take these options into account. A proxy on the delivery path somewhere between the client and the origin Web server temporarily stores requested objects locally and acts as a cache for future requests. The so-called *Soft Caching* [44] approach allows for modification of these resources. Specifically, it allows the proxy to recode images to lower resolution versions and discard original files to save space. Upon a request, the proxy cache might then initially serve the transcoded object with lower image resolution until the original version becomes available.

This possibility adds another level of complexity to the replacement process because a replacement policy no longer only makes a binary decision on whether to completely evict an object or not, but now also needs to evaluate, if storing a copy with reduced quality instead of the original file is beneficial to the overall user experience [7], which is refered to as a *soft decision* [44].

So far, we assumed data to only ever be cached-in when it is demanded and not already in the cache. Under this *demand fetching* model, the replacement policy is the only algorithm of interest [14]. Studies have reported that, due to many single-access requests, the cache hit rate is bound to approximately 50%, even if replacement decisions would always be made optimally by some hypothetical omniscient policy [2], [27].

To lift this bound, document requests must be anticipated and files loaded into the cache before they are eventually demanded. We refer to this as *anticipatory fetching* or *prefetching*. The prediction of future requests should be done accurately and prudently because fetching unnecessary data that will never be needed can significantly increase network traffic and thereby introduce delays [45]. Trace-driven simulations have shown that prefetching data from Web servers into client caches can reduce user-perceived latency by up to 45%, but also doubles the total load on the network [46]. Prefetching is not a cache replacement issue, but the two mechanisms are closely related and cooperatively manage the cache content. When streaming video content over the Internet, preloading sequences that are about to be shown is essential to prevent disruptions in playback. The term "buffering" is often used synonymously to prefetching in the context of VoD applications [39], [40], [47].

Long videos, e.g. movies, are typically consumed linearly from beginning to end, which makes predicting the next requested frames relatively straightforward. The most critical data are, therefore, the early frames of videos. The beginnings of videos are also overall the most popular parts [48], [51]. Sen et al. [47] propose that proxy caches should store a prefix of every audio or video stream, instead of storing the entire object.

Multimedia content popularity is highly dynamic [51], but researchers have identified access patterns that make its prediction feasible. VoD customers are more likely to consume full-length movies in the evening and on the weekend, for example, creating exploitable patterns repeating on a daily and weekly basis [51]. Other research suggests that on video-sharing platforms such as YouTube, popularity patterns differ among categories. Copyright protected material, e.g. a music video, gets a significantly higher percentage of the total views on its first days online, as compared to uncopyrighted videos, e.g. user-generated video blogs, that show steadier request rates on average [49].

5.2. Predictive Least Frequently Used

The traditional LFU policy was presented in Section 3.2.2 and evicts the item that was accessed least often in the past. This section describes *Predictive Least Frequently Used* (P-LFU). P-LFU evicts the item with the lowest predicted number of requests within a specifiable prediction window. For P-LFU to perform well, accurate prediction values are vital. Prediction happens in a separate phase, prior to the actual eviction phase. Famaey, Iterbeke, Wauters and De Turck propose a generic popularity prediction algorithm and find that Web objects can be grouped according to how their popularity evolves

doi: 10.2313/NET-2019-06-1_06

over time. Only four distributions are needed to cover the majority of request patterns for these items [51].

- **Constant** models steady access rates, e.g. for permanently unpopular items.
- **Power-Law** models abrupt steep changes in popularity often seen in multimedia systems.
- **Exponential** models slower changes than power-law and has shown to give the most accurate predictions out of the four distributions when applicable [51], [52].
- **Gaussian** models S-shaped request patterns. Starting off constant, a sudden significant change in popularity appears and then returns to a constant access rate.

Finding the best parameters to fit these four models to the history is a non-linear optimization problem. The authors use the Levenberg-Marquardt algorithm [50] for this purpose. Finally, the distribution with the "best fit", e.g. the one with the smallest *mean squared error* (MSE), is selected and used to make a projection on future request rates.

Famaey et al. conducted simulations on the request traces of the "VoD service of a leading European telecom operator" [51]. 75013 requests were recorded for 4971 different movies. The results indicate that P-LFU can realistically perform approximately 10% better than traditional LFU in terms of hit rate. For accurate predictions, the number of historical data points should not be smaller than 10, however [51].

Further, the algorithm predicted accesses to unpopular movies with significantly higher accuracy than requests for popular items [51].

6. Conclusion and Future Work

There exist plenty of cache replacement proposals beyond what is covered in this paper. Section 3 looked at traditional cache replacement approaches and classified them based on some commonly used factors of cacheable items, like recency and frequency of access that influence the removal process. The Adaptive Replacement Cache (ARC) was presented in Section 4 to demonstrate that improvement over LRU is possible, even without introducing unreasonable overhead. Section 5 described how multimedia Web traffic could pose difficulties for so-called "good enough" replacement strategies in the future. Some unique characteristics of multimedia content were explained and, finally, a prediction-based variant of the LFU scheme that was designed to cope with the challenges of multimedia caching was depicted.

Section 5.1 already touched upon the possibility of prefetching data into the cache before it is actually needed. Prefetching and similar mechanisms should be studied further in the context of Web caching to investigate latency reductions beyond what pure replacement strategies can achieve. Further, transcoding techniques for data reduction might render especially helpful for environments where caches are of smaller size, e.g. mobile systems. How to optimally handle the trade-off between quality and speed that transcoding caches have to deal with is another open issue to be researched.

References

[1] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," Proc. 1997 USENIX Symp. Internet Tech. and Sys., 1997, pp. 193–206.

[2] J. Wang, "A Survey of Web Caching Schemes for the Internet," ACM Comp. Commun. Review, vol. 29, no. 5, Oct. 1999, pp. 36–46.

[3] A. Tanenbaum, "Modern Operating Systems, Third Edition," Prentice Hall, Inc, 2009.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," Proc. IEEE INFOCOM Conf., 1999, pp. 126–34.

[5] H. Bahn, K. Koh, S.H. Noh, and S.M. Lyul, "Efficient Replacement of Nonuniform Objects in Web Caches," IEEE Comp., June 2002, pp. 65–73.

[6] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E.A. Fox, "Removal Policies in Network Caches for World Wide Web Documents," Proc. ACM SIGCOMM Conf., Stanford University, Aug. 1996, pp. 293–305.

[7] S. Podlipnig and L. Boszormenyi, "A Survey of Web Cache Replacement Strategies," ACM Comp. Surveys, vol. 35, no. 4, Dec. 2003, pp. 374–98.

[8] A. Balamash and M. Krunz, "An Overview of Web Caching Replacement Algorithms," IEEE Commun. Surveys & Tutorials, vol. 6, no. 2, 2004.

[9] C. Aggarwal, J. Wolf, and P. Fellow, "Caching on the World Wide Web," IEEE Trans. Knowledge and Data Eng., vol. 11, no. 1, Jan./Feb. 1999, pp. 94–107.

[10] G. Kastaniotis, E. Maragos, V. Dimitsas, C. Douligeris, and D.K. Despotis, "Web Proxy Caching Object Replacement: Frontier Analysis to Discover the 'Good-Enough' Algorithms," Proc. IEEE 15th International Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007, pp. 132–137.

[11] A.K.Y. Wong, "Web Cache Replacement Policies: A Pragmatic Approach," IEEE Network magazine, vol. 20, no. 1, 2006, pp. 28–34.

[12] N. Megiddo and D.S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," Proc. Usenix Conf. File and Storage Technologies (FAST 2003), Usenix, 2003, pp. 115–130.

[13] N. Megiddo and D. Modha, "Outperforming LRU with an adaptive replacement cache algorithm" Computer, vol. 37, no. 4, 2004, pp. 58–65.

[14] N. Megiddo and D. Modha, "One up on LRU," login – The Magazine of the USENIX Association, vol. 28, 2003, pp. 7–11.

[15] N. Megiddo and D. Modha, "System and Method for Implementing an Adaptive Replacement Cache Policy," US Patent 6,996,676, Feb. 2006.

[16] E. Mustain, "The Saga of the ARC Algorithm and Patent," Postgresql General Bits, 2005, http://www.varlena.com/GeneralBits/96.php (accessed September 27, 2018).

[17] M. Chrobak and J. Noga, "LRU is Better than FIFO," Algorithmica, vol. 23, no. 2, 199, pp. 18—185.

[18] G. Rexha, E. Elmazi, and I. Tafa, "A Comparison of Three Page Replacement Algorithms: FIFO, LRU and Optimal," Academic Journal of Interdisciplinary Studies, vol. 4, no. 2, 2015, p. 56.

[19] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance," Proc. ACM Sigmetrics Conf., ACM Press, 2002.

[20] T. Johnson and D. Shasha, "2Q: A Low Overhead High-Performance Buffer Management Replacement Algorithm," Proc. VLDB Conf., Morgan Kaufmann, 1994, pp. 297–306.

[21] D. Lee, J. Choi, J.H. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," IEEE Trans. Computers, vol. 50, no. 12, 2001, pp. 1352–1360.

[22] T. Saemundsson, “An Experimental Comparison of Cache Algorithms,” 2012.

[23] B.D. Davison, “A Web Caching Primer,” IEEE Internet Comput. 5, no. 4, 2001, pp. 38–45.

[24] D. Sleator and R.E. Tarjan, “Amortized Efficiency of List Update and Paging Rules,” Communications of the ACM, vol. 28, no. 2, 1985, pp. 202–208.

[25] J.E. Pitkow and M. Recker, “A Simple yet Robust Caching Algorithm Based on Dynamic Access Patterns,” Proc. 2nd International World Wide Web Conf., 1994, pp. 1039–1046.

[26] K. Rajan and G. Ramaswamy, “Emulating Optimal Replacement with a Shepherd Cache,” Proc. 40th International Symp. on Microarchitecture, 2007.

[27] M. Abrams, C.R. Standbridge, G. Abdulla, S. Williams, and E.A. Fox, “Caching Proxies: Limitations and Potentials,” Proc. 4th International International World Wide Web Conf., 1995.

[28] C.-Y. Chang, T. McGregor, and G. Holmes, “The LRU* WWW Proxy Cache Document Replacement Algorithm,” Proc. Asia Pacific Web Conf., 1999.

[29] J.-M. Menaud, V. Issarny, and M. Banatre, “Improving Effectiveness of Web Caching,” Recent Advances in Distributed Systems, 2000.

[30] M. Arlitt, R. Friedrich, and T. Jin, “Performance Evaluation of Web Proxy Cache Replacement Policies,” Tech. Rep. HPL-98-97(R.1), Hewlett-Packard Company, 1999.

[31] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating content management techniques for web proxy caches,” ACM SIGMETRICS Performance Evaluation Reviews, vol. 27, no. 4, pp. 3–11, March 2000.

[32] A.I. Vakali, “LRU-based Algorithms for Web Cache Replacement,” International Conf. on Electronic Commerce and Web Technologies, 2000.

[33] E. Friedlander and V. Aggarwal, “Generalization of LRU Cache Replacement Policy with Applications to Video Streaming,” 2018.

[34] J. Du, S. Gao, J. Lv, Q. Li, and S. Ma, “A Web Cache Replacement Strategy for Safety-Critical Systems,” Tehnicki Vjesnik, vol. 25, no. 3, 2018, pp. 820–830.

[35] L. Cherkasova and G. Ciardo, “Role of Aging, Frequency, and Size in Web Cache Replacement Policies,” International Conf. on High-Performance Computing and Networking, 2001, pp. 114–123.

[36] N. Young, “On-line caching as cache size varies,” 2nd Annual ACM-SIAM Symp. on Discrete Algorithms, 1991, pp. 241–250.

[37] S. Jin and A. Bestavros, “Sources and Characteristics of Web Temporal Locality,” Proc. IEEE 8th International Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2000, pp. 28–35.

[38] S. Jin and A. Bestavros, “GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams,” Proc. 5th International Web Caching and Content Delivery Workshop, 2000.

[39] A. Dan and D. Sitaram, “Multimedia Caching Strategies for Heterogeneous Application and Server Environments,” Multimedia Tools and Applications, vol. 4, no. 3, 1997, pp. 279–312.

[40] A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, “Buffering and caching in large scale video servers,” Proc. IEEE CompCon, 1995, pp. 217–224.

[41] Wikipedia contributors, “Data compression,” Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Data_compression&oldid=860709233 (accessed September 27, 2018).

[42] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, “Dynamic Adaptation in an Image Transcoding Proxy For Mobile Web Browsing,” IEEE Personal Communications, vol. 5, no. 6, 1998, pp. 8–17.

[43] K.H. Yeung, C.C. Wong, and K.Y. Wong, “A Cache Replacement Policy for Transcoding Proxy,” IEICE Trans. Commun., vol. E87-B, no. 1, 2004, pp. 209–11.

[44] J. Kangasharju, Y.G. Kwon, and A. Ortega, “Design and Implementation of a Soft Caching Proxy,” Computer Networks and ISDN Systems, vol. 30, no. 22-23, 1998, pp. 2113–2121.

[45] M. Crovella and P. Batford, “The Network Effects of Prefetching,” Proc. Infocom’98.

[46] V.N. Padmanabhan and J.C. Mogul, “Using Predictive Prefetching to Improve World Wide Web Latency,” ACM SIGCOMM Computer Communication Review, vol. 26, no. 3, 1996, pp. 22–36.

[47] S. Sen, J. Rexford, and D. Towsley, “Proxy Prefix Caching for Multimedia Streams,” Proc. Infocom’99.

[48] H. Guo, G. Shen, Z. Wang, and S. Li, “Optimized Streaming Media Proxy and its Applications”, Journal of Network and Computer Applications, vol. 30, no. 1, 2007, pp. 265–281.

[49] F. Figueiredo, F. Benevenuto, and J.M. Almeida, “The Tube Over Time: Characterizing Popularity Growth of YouTube Videos,” Fourth ACM International Conf. on Web Search and Data Mining, 2011, pp. 745–754.

[50] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares,” Quarterly Journal of Applied Mathmatics, vol. 2, no. 2, 1944, pp. 164–168.

[51] J. Famaey, F. Iterbeke, T. Wauters, F. DeTurck, “Towards a Predictive Cache Replacement Strategy for Multimedia Content,” Journal of Network and Computer Applications, vol. 36, no. 1, 2013, pp. 21–227.

[52] A. Tatar, M.D. de Amorim, S. Fdida, and P. Antoniadis, “A Survey on Predicting the Popularity of Web Content,” Springer J. Internet Services and Applications, vol. 5, no. 1, 2014, pp. 1–20.

[53] M. Busari and C. Williamson, “On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics,” Proc. IEEE INFOCOM, vol. 3, 2001, pp. 1225–34.

[54] B Krishnamurthy and J. Rexford, “Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement, ” Addison-Wesley Professional, 2001.

Time Sensitive Networking for Wireless Networks - A State of the Art Analysis

Alexander Mildner*
Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Advisor: Fabien Geyer†
Email: mildner@in.tum.de, fgeyer@net.in.tum.de†*

***Abstract*—The trend towards using wireless networking devices for real-time applications is growing, due to the increased flexibility and better cost-efficiency in contrast to wired devices. In this paper we take a closer look at the status of recent research and developments towards enabling Time-Sensitive Networking (TSN) for wireless communication networks. First we provide a brief introduction to the already defined TSN technologies for wired networks. We then discuss some of the main challenges and gaps to overcome, in order to provide a common set of standards for reaching out towards deterministic networking on wireless media. Specifically, we will take a look at some approaches towards accurate clock synchronization via wireless links, including IEEE 802.1AS, which is a key requirement for TSN, in particular for Time Based Scheduling. We will also present some recent efforts on providing deterministic/bounded latencies for various wireless link protocols, e.g IEEE 802.11. Furthermore, we will have a look at different existing solutions, that can enhance reliability and redundancy, by e.g. using additional redundant links, and techniques for managing resource allocation on the End-to-End (E2E) network path.**

***Index Terms*—time sensitive networking, wireless networks, latency, determinism, reliability, 802.1Qbv, 802.1AS, 802.1Q**

1. Introduction

The emerging need for deterministic real-time communication in the industrial automation, automotive and audio/video sectors has pushed recent research and developments in the field of real-time communication and low latency deterministic networks. Today's industrial networks are often dominated by specialized or semi-proprietary wired media fieldbus communication, e.g. ProfiBUS, EtherCAT, CAN-BUS, Ethernet/IP and many more [1]. Although, most of these modern industrial grade Ethernet networks are fundamentally based on the Ethernet standard IEEE 802.3, they are usually closed systems, which differ from vendor to vendor and thus are in most cases not interoperable with each other or common Ethernet networks. A clear downside of these technologies is the lack of flexibility and interoperability capabilities, like the ones provided by the IEEE 802.3 Ethernet or wireless network standards. Especially wireless network infrastructure has advantages in certain deployment scenarios, where wired networks may not be suitable or more expensive to deploy. This results in the need of enabling real-time capabilities and guaranteed deterministic performance bounds, tied into already existing wireless standards, in order to meet the requirements for real-time network applications and furthermore provide an open and common network standardization.

The Institute of Electronics and Electrical Engineers (IEEE), the Internet Engineering Task Force (IETF) and the International Electrotechincal Commission (IEC) have recently proposed new standards in order to introduce deterministic networking to the Ethernet standard. The IEEE Time-Sensitive Networking (TSN) Task Group (TG), formerly named the IEEE Audio/Video Bridging (AVB) TG, has been working on proposing and defining new standardizations and also extending and enhancing already existing ones like IEEE 802.1Q [2] for Quality of Service (QoS), the IEEE 1588v2 Precision Time Protocol (PTP) [3] and IEEE 802.1AS-2011 [4] for accurate clock synchronization in a generalized and optimized manner. With these specifications for a time sensitive, real-time capable and open Ethernet standard, the IEEE TSN TG has defined a set of standards.

In contrast to wired Ethernet based communication, wireless links introduce unreliability, asymmetric channels and latencies, channel interference and signal distortion to the communication path, which makes it hard to provide guarantees for performance characteristics. Thus, wireless links are not suitable for real-time critical and ultra low latency applications by default. Some of the TSN standards already include proposed solutions to solve certain challenges for enabling TSN for wireless networks like adoptions to the IEEE 802.1AS standard, which uses IEEE 802.11 Timing Measurement (TM) [5] for accurate time synchronization over 802.11. Most of the described techniques in the IEEE TSN standards have been defined generically, without further specification on the actual layer 2 network protocol. This allows to extend and adapt the work, by specific requirements or methods to be used on common wireless network protocol standards, such as IEEE 802.11, IEEE 802.15.4 for Wireless Sensor Networks (WSN) and even the upcoming 5G standard. In this paper we take a closer look at the current research on enabling TSN for wireless networks and the main challenges, which need to be resolved.

The remainder of this paper is structured as follows: We first present and refer to some of the most interesting related works on TSN for wireless networks in Section 2. We then provide the current status of the defined TSN standards of the IEEE TSN TG and present

doi: 10.2313/NET-2019-06-1_07

a brief overview of the technologies and methodologies being used in Section 3. After that, we define the most interesting challenges for TSN over wireless networks in Section 4, and furthermore we present the current research efforts and first proposed solutions in order to overcome these problems. We will conclude and summarize the most important findings and give an outlook for future work in Section 5.

2. Related Work

In this section we introduce some recent research studies, which pave the way towards TSN for wireless networks. As this is an ongoing research area, not all remaining challenges have already been approached, thus leaving a lot of open topics for current and future works.

There have been recent efforts in order to define some of the key challenges towards TSN for wireless networks by the Avnu Alliance. The Avnu Alliance is a consortium consisting of professional, automotive, consumer electronics and industrial manufacturing companies, working on defining a common certification for iteroperable TSN standards. The white paper by Steven F. Bush et al. [6], published in January 2018, explores next steps for implementing and deploying TSN methods for industrial wireless networks as a Request for Comments (RFC). The paper covers a proposed roadmap for a seamless integration of Wireless TSN technologies into existing Industrial networks and defines a set of transition phases. It defines characteristical problems of the unreliable wireless networking technologies, which need to be considered, when defining a common Wireless TSN standard. Within each of these transition phases, the author concludes the current status with raising questions regarding open issues, which need to be addressed for the specific underlying radio technology.

The case study of A. Mahmood et al. [7] from April 2017 presented several different approaches for methodologies and protocols for accurate clock synchronization used on IEEE 802.11 wireless networks. The authors examine different existing synchronization methods, e.g. relative vs. absolute synchronization, Hard- vs. Software timestamping methods, built-in 802.11 vs. wired protocols etc., and they are compared against their use on 802.11 based networks. Depending on the use-case, and thus e.g. on the required synchronization accuracy and minimal jitter, the paper suggests to use the PTP protocol with Hardware timestamping for the most accurate method with some additional protocol overhead. Another paper, which has been published in 2011, by A.Mahmood et al. [8] also provided an approach for an implementation of an accurate clock synchronization using PTP with software timestamping for IEEE 802.11 networks. The proposed solution achieved a synchronization accuracy of a few microseconds and jitter below 1μs, by also reducing the protocol overhead by using 802.11 Beacon Frames for timestamp transmission.

Another scientific work, published by A. Nasrallah et al. [9], presented a detailed study on recent developments in Ultra-Low Latency (ULL) networks. This comprehensive case study provides a sophisticated overview of the current efforts on IEEE TSN, IETF Deterministic Networks (DetNet) and 5G technologies for providing real-time and ULL capabilities for Layer 2, Layer 3 and respectively upcoming mobile wireless network standards.

TABLE 1. IEEE 802.1 TSN STANDARDS DEFINED IN [2] WITH TS = TIME SYNCHRONIZATION, BLL = BOUNDED LOW LATENCY, UR = ULTRA RELIABILITY AND RM = RESOURCE MANAGEMENT

	IEEE Std.	Features
TS	802.1AS	Time Synchronization (802.1AS-Rev in draft)
BLL	802.1Qav	Credit Based Shaper
	802.1Qbv	Time Scheduled Traffic
	802.1Qbu	Frame Preemption (also 802.3br)
	802.1Qch	Cyclic Queuing and Forwarding
	802.1Qcr	Asynchronous Traffic Shaping
UR	802.1CB	Seamless Redundancy, Stream Identification
	802.1Qci	Filtering and Policing
	802.1Qca	Path Control and Reservation
RM	802.1Qcc	Stream Reservation Protocol Enhancements
	802.1Qcp	YANG Model for Bridging
	802.1Qcw	YANG Model for Qbv, Qbu, Qci
	802.1CBcv	YANG Model for CB

3. Time Sensitive Networking

In this section we briefly introduce the Time Sensitive Networking technologies that have already been defined for IEEE 802.3 Ethernet and provide an overview of TSN in general. We will define the four key pillars of operating a Time Sensitive Network, which provide the fundamental characteristics of TSN.

One of the main goals of the development of TSN, was to provide an open and standardized technology, not affiliated to any organization or company. The demanding need for real-time capable deterministic networking, which in addition allows interoperability between devices, has pushed the development on TSN forwards. The clear benefits of using TSN over proprietary solutions, are better cost efficiency as common off the shelf (COTS) hardware can be used, establishing network convergence and the technology scales with the Ethernet standard. The TSN technology is based on four basic key concepts, accurate time synchronization, bounded guaranteed low latency, ultra reliability and resource management. For each of these key pillars, the IEEE TSN TG has defined several standards to provide the according functionality (see Table 1). In the following we will have a detailed look at each of these components and the according standards:

1) Time Synchronization (TS): In order to provide a common sense of time, shared between all participating nodes in the TSN network, IEEE 802.1AS [4] provides a generalized PTP (gPTP) profile for accurate time synchronization, using the IEEE 1588v2 PTP protocol [3]. PTP is being used to synchronize the physical hardware clocks (PHCs) of network interface cards (NICs) to a dedicated (Grand-) Master clock on Local Area Networks (LAN), with very high precision. For Ethernet, the accuracy of the offset lies in the sub microsecond range with modern hardware. In contrast to IEEE 1588v2, gPTP provides a generalized and optimized clock synchronization method and provides a set of configuration parameters referred as PTP profile. An accurate clock synchronization is a fundamental premise for most of the IEEE TSN standards, such as IEEE 802.1Qbv for Time based Scheduling. All participating nodes in the TSN-Network must be synchro-

doi: 10.2313/NET-2019-06-1_07

nized, in order to ensure real-time execution and real-time networking.

2) Bounded Low Latency (BLL): One main aspect of real-time applications is message exchange between different nodes in the network on a deterministic low latency basis. Certain traffic classes, such as Cyclic (motion-) control or isochronous (periodic) traffic as defined by A. Ademaj et al. in [12], have strict latency requirements, and thus need a guarantee by the network control for these requirements. For this purpose, the TSN standards provide several methods for ensuring bounded latencies, e.g. 802.1Qbv for Time Scheduled Traffic or 802.1av for a Credit Based Shaper. With 802.1Qbv for example, a pre-defined cyclic time schedule will be deployed throughout the network path. This cyclic time schedule defines for each time slot, which specific traffic classes will be forwarded for the specified amount of time within the cycle. The traffic classes are identified by the IEEE 802.1Q VLAN header, in particular by the priority value. In addition with using time triggered traffic sending (e.g. via SO_TXTIME socket option), time critical traffic can now be sent through the network with constant guaranteed latency, even if there is interfering traffic on the path.

3) Ultra Reliability (UR): Similar to bounded latency, real-time safety-critical applications require high reliability, to ensure their proper functionality. As most of the used traffic types in real-time applications rely on UDP and common Ethernet, there is no built-in reliability mechanism, such as retransmission of lost frames. For this purpose, the TSN standards provide e.g. 802.1CB for Seamless Redundancy and Identification for streams, which replicates frames on a per-frame basis and sends them on 2 (or more) disjoint paths to the target. The duplicate/extra frames will be eliminated on the last network node before the target. This goes hand-in-hand with 802.1Qca for Path Control and Reservation technology, for determining these paths in the network.

4) Resource Management (RM): In a TSN network, each real-time application needs certain network performance requirements, in order to function properly. Another key aspect of enabling TSN is the configuration and management of the available network resources. For this the TSN standards defined both a fully centralized and a fully distributed model for configuration of the network, namely IEEE 802.1Qcc. The trend tends towards using a centralized model, which consists of a Centralized User Configuration (CUC) and a Centralized Network Configuration (CNC) system. These systems are similar to configuration techniques from Software Defined Networking (SDN). The applications request their resource needs at the CUC, which will then trigger the appropriate actions to be taken at the CNC for resource reservation, e.g. for configuring the appropriate cycles for 802.1Qbv on the network path. For the specific data model on the endpoints and the bridges, several YANG [1] models have been designed to be used via the NETCONF or RESTCONF protocol. Figure 2 in section 4 shows a complete model of a TSN network, which uses a centralized configuration model. The Resource Management is currently still under heavy development and certification process.

[1] A data model language for network configuration - see https://tools.ietf.org/html/rfc6020

4. Towards TSN for Wireless Networks

In this section, we discuss the main challenges and gaps for Wireless TSN, which refer to the four fundamental pillars of Time Sensitive Networking, presented in section 3. First, we define the main problems and parameters of a wireless link, that directly affect the fundamental key components of TSN. After that, we will investigate current research efforts on enabling the specific key components of TSN on Wireless Networks.

Wireless Technology has brought up new possibilities and advantages for network communications, in contrast to wired networks. The newly gained flexibility of moving end stations, opened new possibilities to connect devices in a cost efficient way. But in contrast to Ethernet, wireless links introduce several characteristics like unreliability, asymmetric link delay and channels, channel interference and signal distortions from the environment. These properties make the development of suitable standards and implementations for enabling TSN for wireless networks a challenge. A noticeable research effort has already been done, in providing real-time capabilities for IEEE 802.15.4 networks [15], such as Wireless Sensor Networks. But, the overall trend leads towards enabling TSN for IEEE 802.11, due to the higher range and rate, the better interoperability and the improved security mechanisms.

1) Accurate Clock Synchronization: The most important fundamental component of a TSN network is accurate clock synchronization between endstations and also between network devices, such as bridges. For this purpose, we have already introduced the IEEE 802.1AS generalized PTP protocol in section 3. This standard also includes a proposal for clock synchronization for 802.11 in section 12 of [4] by using 802.11 Timing Measurement (TM) [10]. The synchronization procedure is shown in Figure 1.

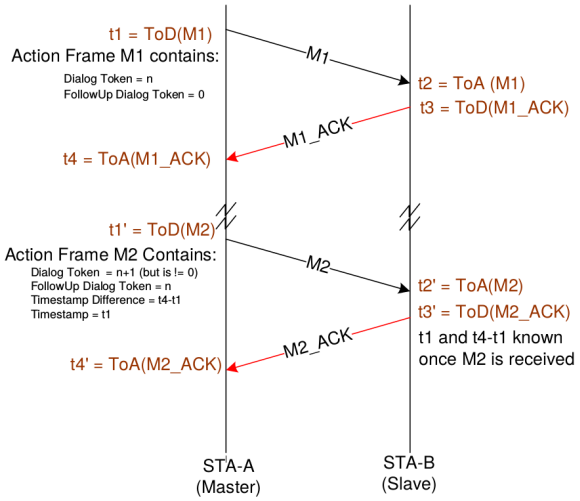

Figure 1. PTP Time Synchronization using TM, ToA = Time of Arrival, ToD = Time of Destination [11]

The biggest difference to the usual gPTP two-step protocol, is that M2 contains information about the previous two step synchronization (specifically $t1$ and $t4 - t1$). With this additional information, channel asymmetries can be detected and accounted into the offset correction calculation on the Slave. The according formulas

doi: 10.2313/NET-2019-06-1_07

for calculating the offset, link delay and the so called $neighborRateRatio$ are as following:

$$neighborRateRatio = (t1' - t1)/(t2' - t2)$$
$$linkDelay = [(t4 - t1) - (t3 - t2)]/2$$
$$timeOffset = [(t2 - t1) - (t4 - t3)]/2$$

The $neighborRateRatio$ is being used, to detect and measure current jitters occurring in unreliable wireless media. These values are then used to synchronize a system clock to the current network time, as the majority of wireless interface cards do not have a built-in PHC. The IEEE 802.1AS-Rev, which is scheduled for release end of 2018, will introduce IEEE 802.11 Fine Timing Measurement (FTM) [5] for an even more accurate clock synchronization, using PTP. Another approach, which does not rely on PTP, for accurate clock synchronization for 802.11 wireless networks, has been proposed by J.-H. Chiang et al. in [14], which uses 802.11 Beacon Frames and the internal Time Synchronization Function (TSF) for synchronizing clocks in a multi-hop wireless network in microsecond range.

2) Bounded Low Latencies: Approaching bounded low latencies is a key challenge for enabling TSN for wireless networks. The main problem is that the unreliability of the wireless media, needs to be measured and monitored, in order to detect e.g. signal quality reduction and current latencies. This is still under heavy development and research. One approach to bypass the unreliability of the wireless media, is to hide the wireless link behind a 802.1Qbv Time Based Scheduler as proposed in [6]. The network nodes need to be synchronized in time to ensure accurate clock synchronization for implementing the time-based cycle. As there do not exist any current implementations of the in Section 4.1 defined clock synchronization methods for wireless networks, one would need to rely on clock synchronization mechanisms for wired ethernet. The cycle needs to take possible retransmissions and maximum latencies of the wireless media into account. Furthermore, the Wireless NIC needs to be capable of controlling the message transmission time and implement the 802.1Qbv scheduler, which is still an open research topic. Although, there have also been approaches to implement a Time-Division Multiple Access (TDMA)-MAC for access control as e.g. by G.-H. Liaw in [13], these techniques cannot meet the requirements of certain use-cases with latencies in the millisecond range e.g. for an industrial real-time application.

3) Providing Reliability: As the IEEE 802.11 standard includes an Automatic Repeat Request (ARQ) mechanism for retransmitting packets that have been lost on transmission, these introduce uncertainties in regards to bounded latency. In Reference [6] the introduction of a redundant system using a wireless network is being proposed, in order to support the redundancy of wired TSN networks, and furthermore gradually introduce wireless networks to industrial IoT use-cases. The Author also states, that 802.11 supports IEEE 802.1CB for Seamless Redundancy and Stream Identification, but that when moving towards enabling TSN for wireless given wireless retransmissions schemes, needs careful design and analysis.

4) Resource Management: In regards of Resource Management, the wireless networks could be used, as a first step, to manage the configuration links between CUC and endpoints and CNC and network devices. In Figure 2 the complete structure of a decentralized configuration based TSN network can be seen. As suggested in [6], the dotted lines could be deployed as wireless links, as this is only for control not for operation. One advantage would be that there could be mobile Access terminals, which would allow to control the applications via wireless networks. Furthermore, replacements of equipment could be simplified, as no physical wiring needs to be changed. Of course, once there are suitable TSN standards to operate a TSN network over wireless media, there will be the need of certain standards for a configuration model specification for wireless bridges and endpoints. These would have different configuration parameters as wired stations. Furthermore there would be the need of an analyzing or monitoring instance, which would collect and analyze data about the current status of the wireless channels, in order to ensure resource availability.

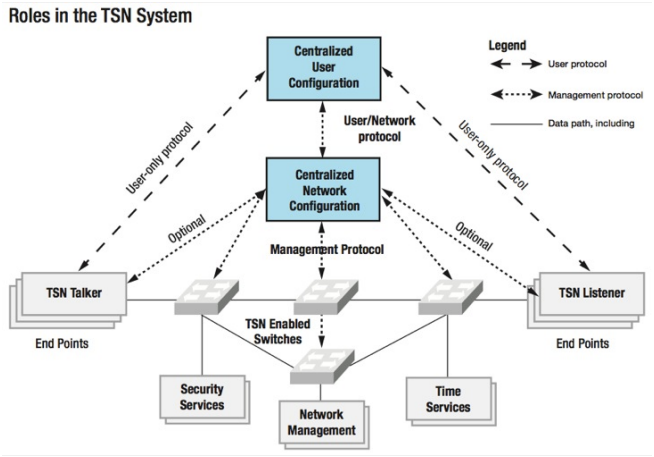

Figure 2. TSN Resource management [5]

5. Conclusion and Future Work

In this paper we showed only an excerpt of the current efforts to enable TSN on wireless networks. We discussed the main challenges for providing accurate clock synchronization, reliability, deterministic or bounded latencies and resource management and provided an insight to current developments, in order to approach some of them. The current status of TSN for wireless is in an early development stage and there are still many open topics to address for an open standardization. Accurate clock synchronization for wireless networks still needs further investigations and improvements, especially improved hardware support for 802.1AS for wireless NICs. Another future work would be to investigate the performance improvements of clock synchronization using the upcoming 802.1AS-Rev standard, in comparison with 802.1AS and 802.11 Beacon Frame methods. In terms of bounded latencies, further developments of time based packet transmission, as done for Ethernet, need to be considered for future work. Furthermore, there is the need for a sophisticated solution for measuring and monitoring quality parameters of the wireless links, in order to make guarantees for resource management.

doi: 10.2313/NET-2019-06-1_07

References

[1] 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,” in IEEE Industrial Electronics Magazine, vol. 11, no. 1, pp. 17-27, March 2017.

[2] “IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks,” in IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014) , vol., no., pp.1-1993, 6 July 2018.

[3] IEEE Std. 1588-2008, “IEEE Standard for A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.”, IEEE Std., 2008.

[4] IEEE, “802.1AS-2011 - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks”, IEEE Std., 2011.

[5] “IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” in IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012) , vol., no., pp.1-3534, 14 Dec. 2016.

[6] S. F. Bush et al. “Industrial Wireless Time-Sensitive Networking: RFC on the Path Forward,” [White paper], Retrieved on Sep. 18, 2018, from AVNU Alliance Community, https://avnu.org/wp-content/uploads/2014/05/Industrial-Wireless-TSN-Roadmap-v1.0.3-1.pdf.

[7] A. Mahmood, R. Exel, H. Trsek and T. Sauter, “Clock Synchronization Over IEEE 802.11 - A Survey of Methodologies and Protocols,” in IEEE Transactions on Industrial Informatics, vol. 13, no. 2, pp. 907-922, April 2017.

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

[9] A. Nasrallah et al. , “Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research,” in IEEE Communications Surveys & Tutorials, Sep 2018.

[10] K. B. Stanton, “Distributing Deterministic, Accurate Time for Tightly Coordinated Network and Software Applications: IEEE 802.1AS, the TSN profile of PTP,” in IEEE Communications Standards Magazine, vol. 2, no. 2, pp. 34-40, JUNE 2018.

[11] K. B. Stanton, “Tutorial: The Time-Synchronization Standard from the AVB/TSN suite IEEE Std 802.1AS-2011 ,” Retrieved on Sep. 29, 2018, from IEEE, http://www.ieee802.org/1/files/public/docs2014/as-kbstanton-8021AS-tutorial-0714-v01.pdf.

[12] A. Ademaj, “Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types,” Retrieved on Sep. 29, 2018, IIC Consortium, https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Traffic_Whitepaper_20180418.pdf.

[13] G. Liaw and Y. Yeh, “A TDMA Medium Access Control Mechanism for IEEE 802.11-Based Wireless Networks,” 2011 Fifth International Conference on Genetic and Evolutionary Computing, Xiamen, 2011, pp. 61-64.

[14] J.-H. Chiang and T. Chiueh, “Accurate clock synchronization for IEEE 802.11-based multi-hop wireless networks,” 2009 17th IEEE International Conference on Network Protocols (2009): 11-20.

[15] Feng Chen, T. Talanis, R. German and F. Dressler, "Real-time enabled IEEE 802.15.4 sensor networks in industrial automation," 2009 IEEE International Symposium on Industrial Embedded Systems, Lausanne, 2009, pp. 136-139.

doi: 10.2313/NET-2019-06-1_07

Measuring TCP Performance Metrics with Bro

Leonhard Stemplinger, Simon Bauer*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: leonhard.stemplinger@tum.de, bauersi@net.in.tum.de

***Abstract*—The Transmission Control Protocol (TCP) is one of the most widely used networking protocols. The ability to accurately measure the performance of a TCP connection is important for identifying network problems. In this paper, we present an implementation of three TCP performance metrics: the inter-arrival time of acknowledgements, the time-series of retransmissions and the retransmission score. We compute these metrics using the Bro network analysis framework.**

***Index Terms*—tcp, bro, retransmissions, network monitoring**

1. Introduction

TCP is the most common transport layer protocol, with over 90% of internet traffic transmitted over TCP [14]. This makes analysing TCP connections important for network operators, as performance problems could impact the majority of users. One indicator of possible network problems is a high rate of retransmissions, as this indicates a high rate of packet loss. We present Bro scripts to measure several TCP performance metrics for both live connections and trace files. These metrics are the inter-arrival times of acknowlegments, a time-series of retransmissions, and the retransmission rate. They were defined by Siekinnen et al. [2] as part of a root cause analysis framework. We extend their work by describing a possible implementation in detail. Additionaly, our implemetation is able to analyse live traffic, while [2] is limited to trace files.

In this paper, we will first summarize important aspects of the TCP protocol and the Bro network monitor in Section 2. We continue by defining the TCP perfomance metrics implemented for this work in Section 3 and listing other works dealing with related topics in Section 4. In Section 5 we describe an implementation of these metrics in Bro scripts and present the results computed by the scripts in a test in Section 6. We show possibilities for further work in Section 7 and provide access to the Bro scripts and conclude the paper with Section 8.

2. Technical Background

2.1. TCP Protocol

TCP is a transport layer protocol. It relies on the underlying Internet Protocol to transfer packets to their destination. TCP aims to provide a reliable connection even over unreliable network infrastructure [9]. To achieve this, a TCP connection must be established using a three-way-handshake, and both endpoints must keep status information over the lifetime of the connection. Additionally, damaged or lost packets must be detected and retransmitted.

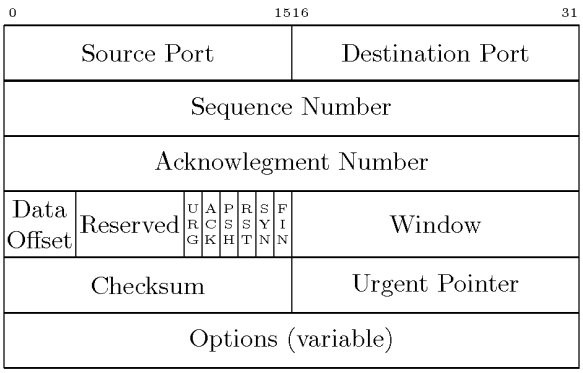

Figure 1. The TCP header [9]

The most relevant part of TCP for this paper is the acknowledgement mechanism. TCP packets carry a sequence number, which is incremented for every byte sent by an endpoint. The receiver acknowledges correctly received data by incrementing the responses acknowledgment number. A packet with set ACK flag and an acknowledgment number n indicates that all data up to and including sequence number $n-1$ has been received by the connection partner.

Using this mechanism, TCP attempts to detect and retransmit lost packets. Generally, TCP will assume a packet was lost if it is not acknowledged within a certain timespan, or when receiving multiple acknowlegments for previous packets. Details vary between TCP implementations [6]. If a TCP endpoint receives non-contiguous data, meaning data has been lost, but a later packet was received successfully, the selective acknowledgment (SACK) option can be used to notify the sender of the received data and avoid unneccessary retransmissions [10] [12].

2.2. Bro

Bro [8] is a free, open source network monitoring tool. It was originally published by Paxson in 1999 [1]. Bro can process both trace files and live traffic.

Bro is divided into two main components. The Event Engine generates events based on the observed network

doi: 10.2313/NET-2019-06-1_08

activity. The Policy Script Interpreter executes scripts in response to the generated events. Bro scripts are written in the Bro scripting language. Each Bro script defines a number of event handlers that are called whenever the corresponding event is triggered. While Bro comes with a large library of scripts for traffic analysis, it can also be extended with custom scripts.

We mainly chose Bro because the abstraction provided by the event system made development of our traffic analysis scripts easier and faster. Bro scripts do not need to perform low level traffic processing such as separating TCP and non-TCP traffic, detecting the TCP connection a packet belongs to or extracting header information from raw packets, as this is handled by the event engine. Additionally they work on live traffic as well as trace files without modifications.

3. Metrics

This paper describes a solution to measure three of the metrics defined in [2] using Bro scripts. This section describes the selected metrics. All metrics are measured separately for each direction of a TCP connection.

3.1. Inter-Arrival Times of Acknowledgements

Each endpoint of a TCP connection acknowleges correctly received packets. For every acknowlegement, we record its' arrival time, the number of acknowledged bytes and the interval since the last observed acknowledgment. Packets that do not advance the acknowledgement number are not recorded. The inter-arrival times can be used to estimate the capacity of the connections' network path [2].

3.2. Retransmission Metrics

TCP detects lost packets by monitoring acknowledgments and retransmits those packets. For this paper, retransmitted packets are defined as packets with a sequence number lower than or equal to the highest previous sequence number. The definition in [2] additionally demands an IPID higher than all previous packets to eliminate false positives caused by out of order packets. However the IPv4 specification has since been updated to allow arbitrary ID values for non-fragmented packets [3]. Furthermore, non-fragmented IPv6 packets do not carry any ID value [16]. Packets without a payload (i.e. pure ACKs) are not included in the analysis, as they do not advance the sequence number.

We measure two metrics to analyse retransmissions. The timestamp and payload size of each retransmitted packet are recorded to create a time-series of retransmissions. Additionaly, we keep track of the amount of data retransmitted, and the total amount of data transmitted. The ratio of retransmitted data to total data is the retransmission score. A high retransmission score generally correlates to a high rate of packet loss. If this occurs frequently, it can indicate a network problem.

4. Related work

The TCP performance metrics implemented for this paper are a subset of those described by Siekinnen et al. [2]. They define several other metrics, as well as a procedure to determine limiting factors for a connections' throughput based on these metrics. However, they do not provide implementation details.

There have been many other works studying TCP retransmissions. Examples include Pentikousis et als. [5] analysis of aggregate retransmission rates among a large number of connections. Rewaskar et al. [6] and Jaiswal et al. [7] present methodologies to classify out-of-order packets into retransmissions and reordered packets.

The TCPRS Bro plugin by Swaro [13] extends Bro with additional events for reordered and retransmitted packets. While the implementation for this paper is written entirely as Bro scripts, TCPRS adds a new analyzer to the Bro event engine.

Most publications related to Bro deal with network security topics, such as intrusion detection, and not network performance. One exception is Sargent and Allmans analysis [15] of the limiting factors for very high bandwith (1 Gb/s) residential fiber connections. They report that current TCP implementations do not use such a connection efficiently, as receiver window sizes limit data transmission to a far lower rate.

5. Implementation

This section describes the Bro scripts that implement the metrics described above.

5.1. Common components

This section describes some patterns that occur in both scripts.

5.1.1. State Information. Both scripts need to keep information about previous packets observed in each direction of each connection. For this purpose, both scripts include two tables to store this information, one for each direction of a connection. A Bro script table is a key-value store. In this case, the keys are the unique IDs (uids) generated by bro for each connection, and the values are records, a data type similar to C structs, that hold information about the connection.

Records are added when observing a packet that does not belong to a previous connection. They are deleted when Bro deletes the connection by handling the connection_state_remove event.

5.1.2. Analyzing new TCP packets. To monitor TCP connections, both scripts register a handler for the tcp_packet event, which is triggered for every TCP packet . The handler receives a connection record, a flag indicating by which endpoint the packet was sent and the values of various TCP header fields. The connection record has a large number of fields which hold information about aspects of the connection. For our scripts, only the connection uid is needed.

5.1.3. Logging. Both scripts record their results in Bro log files (acks.log, retransmission_series.log and retransmission_scores.log). While the result format is different, the first three items of each entry are the same in all logs:

doi: 10.2313/NET-2019-06-1_08

- Connection UID: For cross-referencing with other Bro logs
- Timestamp
- from_orig: A boolean flag that indicates which direction of the connection the entry concerns.

5.2. Inter-arrival times of Acknowledgements

The acknowledgment script stores two numbers for each direction of a connection: The highest ack number of the observed packets, and the timestamp of that packet. For each new packet, the script first checks wether the packet is relevant for the time-series. Packets without a set ACK flag, as well as packages whose acknowledgment number is lower than or equal to the highest number recorded for the same direction are filtered out. If the packet passes this filter, the stored information about the corresponding connection is updated, and a log entry is created. In addition to the fields listed in section 5.1.3, it contains the packets' acknowledgment number, the number of bytes acknowledged (i.e. the difference between the new ack number and the previous one), and the interval since the last acknowldegment.

5.3. Retransmission Metrics

For each direction of a connection the retransmission metrics script stores the maximum sequence number, the number of bytes previously retransmitted and the total number of bytes transmitted, including retransmissions. It also stores wether these values have changed since the last time the retransmission score was computed. Whenever a new packet with a payload is received, the script updates the amount of transmitted data. It also determines wether the packet is a retransmission by comparing its' sequence number to that stored for the connection. If it is not a retransmission, the new maximum sequence number is saved. If it is, the number of retransmitted bytes is updated, and an entry is added to retransmission_series.log.

The information gathered this way is used to calculate the retransmission score. The script defines a new event score_log_trigger and schedules it to trigger every 0.1 seconds. In a handler for this event, the script updates retransmission_score.log. For each direction of each connection, a new log entry is created, if there was a packet observed in this direction since the last log entry.

6. Evaluation

The scripts were tested on a packet capture of the download of part of the Bro documentation. Both Figure 2 and Figure 3 show results for the same connection.

Figure 2 shows that over this connection, data was sent almost exclusively from the responder to the originator. The only packets acknowledged by the responder are those necessary for the TCP and TLS handshakes and for the TCP teardown. After the data transfer, there was a period of inactivity for about two seconds before the connection closed. As seen in Figure 3, the connection experienced a spike in retransmissions after approximately two seconds without retransmissions. Afterwards, the retransmission score decreased again as more data was transmitted.

```
type ack_time: record{
        ack: count;
        timestamp: time;
};

global last_orig_acks: table[string] of
↪  ack_time;
global last_resp_acks: table[string] of
↪  ack_time;

event tcp_packet(c: connection, is_orig:bool,
↪  flags: string, seq: count, ack: count,
↪  len: count, payload: string){
        if ("A" !in flags){
                return;
        }
        local timestamp=network_time();
        local first_ack: bool;
        local last_acks=last_resp_acks;
        if (is_orig){
                last_acks=last_orig_acks;
        }
        first_ack=(c$uid !in last_acks);
        if (first_ack){
                last_acks[c$uid]=ack_time($ack=ack,
                ↪  $timestamp=timestamp);
        }
        if (ack <= last_acks[c$uid]$ack &&
        ↪  !first_ack){
                        return;
        }
        handle_ack(last_acks[c$uid], ack,
        ↪  timestamp, is_orig, c, first_ack);
        last_acks[c$uid]=ack_time($ack=ack,
        ↪  $timestamp=timestamp);
}
```

Script 1: The acknowledgement scripts' tcp_packet handler

```
event score_log_trigger(){
        local ts=network_time();
        log_scores(orig_info, T, ts);
        log_scores(resp_info, F, ts);
        schedule 0.1sec {score_log_trigger()};
}
```

Script 2: Retransmission score logging

7. Future Work

Currently, our script uses a very simple retransmission detection mechanism. The precision of of the retransmission metrics could be improved by a more sophisticated mechanism that is able to separate true retransmissions and reordered packages, removing false positives.

As mentioned in section 4, not all of the metrics described in [2] were implemented for this work. The remaining metrics could be implemented in similar Bro scripts. This would enable the use of the root cause analysis procedure described in [2].

However, Bro scripts might not be the best tool for calculating these metrics, in particular for higher bandwidth connections. The Bro documentation [4] warns of

doi: 10.2313/NET-2019-06-1_08

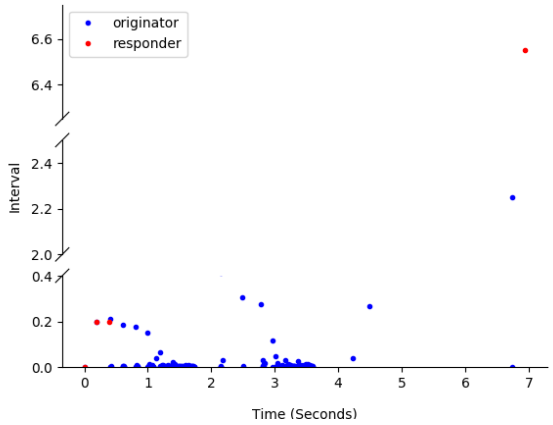

Figure 2. A plot of the intervals between acknowledgments over the duration of a connection.

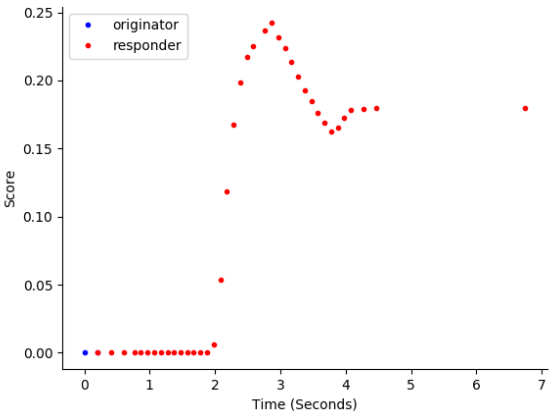

Figure 3. A plot of the retransmission score over the duration of a connection

the performance impact of handling the tcp_packet event. While no performance problems were observed in our tests, these were done with fairly small trace files and results could change in higher load situations. Additionally, the tcp_packet event does not expose the size of the TCP window, which is needed for some metrics. To access this value, a script needs to use the more general new_packet event. This could add to to any performance problems, as new_packet is triggered more frequently than tcp_packet. One solution to possible performance problems could be moving packet classification from the tcp_packet handler to a custom Bro traffic analyzer. As Bros' traffic analysis layer is built in C++, it would likely run far faster than the interpreted scripts.

8. Conclusion

In this paper, we summarized the TCP acknowlegment and loss recovery mechanism and introduced three performance metrics based on this mechanism: the inter-arrival times of acknowledgments, the time-series of retransmissions and the retransmission score. We described Bro scripts that compute these metrics for both trace files and live traffic and tested them on real-world internet traffic.

Both of our scripts are available at [11]. For instructions on how to set up Bro and run custom Bro scripts, please refer to the Bro documentation [8].

The implemented metrics are of limited use on their own. However, this work shows the feasibility of using Bro for TCP performance analysis. Combined with scripts for other metrics, the Bro scripts shown here could form part of a more sophisticated measurement toolkit.

References

[1] Vern Paxson, Bro: A System for Detecting Network Intruders in Real-Time, Computer Networks, 31(23-24), pp. 2435-2463, 1999

[2] Matti Siekkinnen, Guillaume Urvoy-Keller, Ernst W. Biersack, Denis Collange, A root cause analysis toolkit for TCP, Computer Networks, Volume 52, Issue 9, Pages 1846-1858, 2008

[3] J. Touch, Updated Specification of the IPv4 ID Field, RFC 6864, 2013

[4] Documentation of the Bro_TCP.events script, www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html

[5] Kostas Pentikousis, Hussein Badr, Asha Andrade, A Comparative Study of Aggregate TCP Retransmission Rates, International Journal of Computers and Applications, 32:4, 435-441, 2010

[6] Sushant Rewaskar, Jasleen Kaur, F. Donelson Smith, A Passive State-Machine Approach for Accurate Analysis of TCP Out-of-Sequence Segments, ACM SIGCOMM Computer Communication Review, Volume 36 Issue 3, Pages 51-64, 2006

[7] Sharad Jaiswal, Gianluca Iannacone, Cristophe Diot, Jim Kurose, Don Townsley, Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone, IEEE/ACM Transactions on Networking, Volume 15 Issue 1, Pages 54-66, 2007

[8] The Bro Network Security Monitor, www.bro.org

[9] Transmission Control Protocol, RFC 793, 1981

[10] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, TCP Selective Acknowledgement Options, RFC 2018, 1996

[11] github.com/lstemplinger/bro-tcp, Commit c4b6dc9

[12] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, Y. Nishida, A Conservative Loss Recovery Algorithm Based on Selective Acknowledgement (SACK) for TCP, RFC 6675, 2012

[13] James Swaro, TCP Retransmission and State Analyzer plugin for the Bro-IDS framework, github.com/jswaro/tcprs

[14] DongJin Lee, Brian E. Carpenter, Nevil Brownlee, Media Streaming Observations: Trends in UDP to TCP Ratio, International Journal on Advances in Systems and Measurements, vol 3 no 3&4, 2010

[15] Matthew Sargent, Mark Allman, Performance within a fiber-to-the-home network, ACM SIGCOMM Computer Communication Review, Volume 44, Issue 3, pages 22-30, 2014

[16] S. Deering, R.Hinden, Internet Protocol Version 6 (IPv6) Specification, RFC 8200, 2017

doi: 10.2313/NET-2019-06-1_08

Open vSwitch Configuration for Separation of KVM/libvirt VMs

Jonas Andre, Johannes Naab*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: andre@in.tum.de, naab@net.in.tum.de

***Abstract*—Virtual machines are very useful tools to provide configured computers for a lecture. These machines are usually connected via a layer two network to the Internet. As the virtual machines cannot be trusted because they can send arbitrary frames, the network and other virtual machines need to be protected of network attacks. This paper presents a solution and implementation to improve the security of libvirt virtual machines connected via an Open vSwitch. The outcome is an Open Flow implementation which prevents virtual machines from attacks like spoofing, Denial of Service, and Man in the Middle attacks.**

***Index Terms*—Open vSwitch, libvirt, Open Flow, virtual machine security, layer 2 attacks**

1. Introduction

Lectures like *Grundlagen Rechnernetze und Verteilte Systeme* and *Advanced Computer Networks* at TUM provide virtual machines (VMs) to their students. On those machines they perform practical experiments. Using VMs is comfortable as it provides each student the same configuration where the experiments work out of the box.
For the experiments, the machines need to be connected to the Internet. This is done via a single layer two network connecting all VMs with their gateway. As all machines are within a single layer two network, there are attacks which could not only harm the VM of a student, but all VMs within the network.
This work deals with security problems of multiple VMs within one layer two network. The paper specializes on VMs virtualized by the Linux KVM module, configured by using the libvirt API, where the network is managed via the Open vSwitch module.
As discovered in previously running environments with 900 VMs, the CPU workload should be addressed. ARP requests need to be processed at all machines parallel. With this number of VMs and a high number of ARP requests, the CPU load of the host grew significantly. A solution to prevent ARP broadcasting is also presented.
The paper is structured as follows. In Section 2, popular (layer two) attacks which are relevant for the used scenario are considered. Requirements to prohibit those attacks are determined. Afterwards, the section describes the CPU workload problem related to the network. In Section 3, the current implementation is described and evaluated. The new implementation for the libvirt/KVM scenario with Open vSwitch is described in Section 4. The conlusion in Section 5 evaluates the implemented rules and shows investigations for future work.

2. Security Issues in Layer 2 Networks

In this section, relevant security issues within layer two networks are explained. A short description of possible attacks is given. It also shows the theoretical possibilities to prevent those attacks. Authenticity is not given for sender and receiver addresses within layer 2 network frames. Consequently, there are several spoofing attacks on which a VM can send with fake IP and MAC addresses.

MAC Spoofing. If a VM wants to hide what packets it is sending, it can use MAC spoofing. In this attack the sender machine does not send an Ethernet frame with the source MAC of its own interface. Instead, a fake MAC address or the address of another machine within the network is used. With this it is not possible anymore to track the sender of the frame. To be able to determine the true sender of a frame we must ensure that the VMs can only send frames with their own (predefined) MAC address. To prevent MAC Spoofing the following requirement needs to be fulfilled.

Req.1 VMs can only send Ethernet frames from their configured MAC address

IP Spoofing. Another spoofing attack is IP/IPv6 spoofing. Here, the attacker uses a fake IP address to hide its identity to hosts outside of this layer two network. The problem is that attacks on hosts outside of the layer two network identify the attacker by its IP address. This address can be tracked back to the IP network of the VMs. If the address was spoofed, it is not possible to find the attacker within the network. This should be possible, as the responsible VM for such an attack needs to be found. To achieve this two requirements need to be met:

Req.2 VMs can only send IPv4 frames from their configured IPv4 address

Req.3 VMs can only send IPv6 messages from their configured IPv6 addresses

ARP Spoofing. A well known attack which may result in a DoS or a MitM is the so called ARP spoofing [1]. ARP is designed to find the MAC address of a machine within a layer two network by its IPv4 address [2]. On an ARP request resolving a specific IP address the machine with the configured IP address should answer with a reply containing the MAC address of its interface on which the IP address is configured. However, any machine could answer on an ARP request. If a machine answers to an ARP request which is not meant for it, future IP packets will be routed to the spoofing machine instead to its

doi: 10.2313/NET-2019-06-1_09

correct destination. The following requirement protects the network from APR spoofing:

Req.4 VMs can only answer with ARP requests containing their configured IP address

IPv6 Router Spoofing. A Router Solicitation is a message that is sent from a host to find routers within its layer two network [3]. Listening routers answer with router advertisements which include information about the network configuration. VMs can manipulate the interface configuration of other VMs. This is possible by sending Router Advertisements by themselves. As this can lead to DoS and MitM attacks [4], the messages must be forbidden.

Req.5 VMs are not allowed to send Router Advertisements

Neighbor Discovery Spoofing. In IPv6 NDP also takes the place of ARP in IPv4. Spoofing is here possible in the same way as it is described before. To prevent spoofing, the machines are only allowed to answer to Neighbor Solicitations meant for them. This also addresses a problem with Stateless Address Auto Configuration (SLAAC) [5]. With SLAAC IPv6 interfaces generate their own IP address. In our setup SLAAC is used to configure link local IP addresses. One step is to check whether the IP address generated is already used. For this, a Duplicate Address Detection (DAD) message is sent. This is a Neighbor Solicitation message for the generated address. If no one answers to this message, the address can be assigned to the interface. A typical attack in DAD is a machine answering to these DAD messages although it is not configured with this address. This leads to a DoS as the VM cannot generate an IPv6 address.

Req.6 VMs can only send Neighbor Advertisements containing one of their configured IPv6 address

ICMP Redirect. Another possible attack is the abuse of the Redirect message [4]. This message is usually used to inform clients that a router which received the packet knows a better route to the destination IP. By sending such messages, the VMs can be manipulated to send packets to other VMs instead of the gateway.

Req.7 VMs are not allowed to send the Redirect messages

Broadcast Flooding. As ARP requests are broadcasted in Ethernet networks [2], every VM gets all ARP request. Every machine needs to process the request to decide whether it needs to response to the request or not. Hundreds of VMs processing ARP requests at the same time leads to high CPU consumption. The reason for the high CPU consumption lies in the Spectre and Meltdown security fixes. As the VMs have to be loaded and unloaded every time to prevent reading uncleaned memory of the other machines. This needs to be done for every incoming ARP request. The workload for loading and unloading the machines is high. There should be a solution such that the ARP requests do not need to be sent to all machines. This is possible because all IP addresses of the virtual machines are known in advance.

Req.8 Prevent ARP broadcasting by sending ARP requests directly to the correct machine

DHCP. The IP addresses of the VMs are known in advance. Nevertheless, a DHCP server is useful to assign IP addresses to the machines. An attacker can fake a DHCP server on a virtual machine by answering packets destined for the original DHCP server or sending DHCP offers to other clients [6]. This should be denied as this may lead to DoS or MitM for other VMs. DHCP server messages can be dropped by filtering on the DHCP server UDP source port.

Req.9 VMs are not allowed to send DHCP server messages

3. Current State

In this section, the setup of the network connecting the VMs is described. It also shows how the machines are created and configured as this is the basis on which the countermeasures against network attacks can be piggybacked. At the end of this section the defenses installed currently are evaluated.
There is one Open vSwitch which connects all VMs with the gateway to the Internet. The Open vSwitch with name `vm-switch` is shown in Figure 1. Traffic from and to the

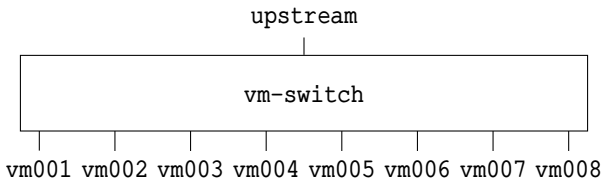

Figure 1: The switch and its ports

Internet goes through interface `upstream`. All VMs are connected via an interface `vmXXX` where `XXX` is the number of the VM.
The VMs are created via the command `virt install`. A script called `create-student.sh` generates one or multiple VMs for a user. To be able to access the generated information, name and address(es) of the interface(s) are encoded within the *metadata* option of the `virt install` command.
The necessary security rules are created via Open Flow. A script for generating these rules is triggered by a QEMU hook. QEMU is the machine emulator on which KVM is based on. On every start of a VM the – within the `virt install` defined – *metadata* is parsed. With this information the script finds the port identifier of the Open vSwitch on which the VM is connected to `vm-switch`. The MAC and IPv4 address of the interface are also extracted from the *metadata* file. With information about port ID, MAC address, and IPv4 address, Open Flow rules are installed on the switch. It is intended to create rules to prevent MAC and ARP spoofing. The following rules are created with the *ovs-ofctl create-flow* command within the QEMU hook. To provide better readability, the `in_port=$port` which is part of every rule is omitted.

```
1 dl_src=$mac priority=40 action=normal
2 dl_src=* priority=39 action=drop
3 arp arp_sha=$mac nw_src=$ip4 priority=40
    ↪action=normal
4 arp arp_sha=$mac nw_src=* priority=39 action=
    ↪drop
```

Listing 1: Installed rules

doi: 10.2313/NET-2019-06-1_09

The rules shown in Listing 1 are written in Open Flow syntax. They are intended to work the following way:

- Allow frames from the given port where the MAC address matches the configured MAC address of the machine
- Drop all frames with another MAC address
- If the packet is an ARP response, allow only responses of the configured IP and MAC address
- Drop all other ARP responses

During the implementation of security features (explained in Section 4) within this work, it was detected that these rules do not work in the intended way. The problem lies in the first rule. As it has the same priority as rule number three, it is not deterministic which of those rules is considered at first [7]. Therefore, if an ARP response with the configured source MAC address and a spoofed IP address is sent the action depends on the chosen rule. If the third rule is used, everything is fine. In case the first rule is used the switch determines a correct sender MAC address and performs the action `normal`. This means the packet is processed like it is a normal unconfigured switch, i.e., it is sent to the given destination. With this ARP spoofing is possible. An attacker using this ARP spoofing attack can act as a Man in the Middle for packets which are destined for the spoofed IP address. This is the case as all packets are first sent to the attacker because the sender expects the IP address at the attacker.
At the moment the following issues are not considered

- Defenses against attacks based on IPv6
- DHCP attacks
- DoS by spamming broadcasts
- DoS by ARP request which may lead to CPU overconsumption as all machines have to process the packets
- Cleanup of old rules when a VM is destroyed

4. Implementation

This section is about how the issues addressed in Section 2 are implemented. The changes in the script for creating the students VMs and how the security relevant rules are installed are pointed out.
The script for creating the VMs was updated to handle different new functionalities. Configured IPv6 addresses of the machines are now also written to the *metadata* of the machines. This is necessary for the automatic creation of security rules.
These rules are created via a hook which is triggered by the QEMU creation of the VMs. It runs during step *start* and phase *begin*. The basic functionality of the script is to create new security rules for each VM and delete those rules when the machine is deleted or shut down. To implement the security rule in the Open vSwitch, the `ovs-ofctl add flows` command is used.
When the script is started, the data written to the *metadata* file is parsed. As only the information about the different interfaces MAC, IPv4, and IPv6 global unique addresses are given, the link local addresses of the interfaces need to be derived. With the specified interface name in the *metadata* of the VM, the Open vSwitch port ID of the machines gateway interface can be found. This port ID is necessary to generate input-dependent flow rules on the switch. For convenience and an easy way to delete the created rules, rules are tagged with an unique identifier (called `cookie`) of the machines. This is the MAC address of the interface connecting the VM to the switch.

table0	table1	table2	table3
MAC spoof	IP spoof	Direct ARP	ICMP spoof
ARP	ARP spoof		
	DHCP spoof		

Figure 2: The Open Flow tables of vm-switch

Open vSwitch manages its OpenFlow rules with different tables. Each table consists of several rules with different priority. A table needs to have an entry with priority `0`. This rule is the default rule (table miss rule). In an Open vSwitch with default configuration there exists only one table called **table0**. This has only the default rule with action `normal` which indicates the switch should handle the frame like a typical switch. Beside `normal` there exist more actions like `drop` (dropping the frame), `goto_table:X`, where `X` is the id of the table in which the frame should be tested next, and `X` which sends the frame to port ID `X` [7].
The implemented rule set is defined in four tables. An overview of the structure can be seen in Figure 2. It shows which table handles the specific requirements.
The following source code shows the installed rules of the different tables. The `cookie` is not represented below for providing better readability. If no `in_port` is specified within a rule, it means this rule is only applied on the incoming port where the VM is connected to. The entry table **table0** prevents MAC spoofing and initiates the ARP optimization.

```
1  dl_src=$mac priority=40 action=goto_table:1
2  priority=39 action=drop
3  in_port=* arp priority=1 action=goto_table:2
4  in_port=* priority=0 action=normal
```

Listing 2: Installed rules in `table0`

The first rule of Listing 2 defines that frames coming from the connected machine which are sent with the correct MAC address are processed further in **table1**. The next rule (it has lower priority) drops all packets coming to the switch which are not sent from the defined MAC address. It matches against all MAC addresses. However, the correct address is already tested in the previous rule. With these two rules, Req. 1 as specified in Section 2 is met. All packets that do not enter the switch from a virtual machine (interface `upstream`) are first matched against the third rule. This rule does not specify an `in_port`. It tests if the frame is an ARP packet. Then it is further processed in the ARP optimization table (**table2**). All other packets from `upstream` match against the table miss entry which defines to process the frame with action `normal`.

```
1  arp arp_sha=$mac arp_spa=$ip4 priority=40
       ↪action=table2
```


doi: 10.2313/NET-2019-06-1_09

```
2  udp udp_src=67 priority=39 action=drop
3  ip nw_src=$ip4 priority=38 action=normal
4  icmp6 ipv6_src=$ip6 priority=37 action=table3
5  icmp6 ipv6_src=$eui64 priority=36 action=
       ↪table3
6  icmp6 ipv6_src=:: priority=35 action=table3
7  udp6 udp_src=547 priority=34 action=drop
8  ipv6 ipv6_src=$ip6 priority=33 action=normal
9  ipv6 ipv6_src=$eui64 priority=32 action=
       ↪normal
10 in_port=* priority=0 action=drop
```

Listing 3: Installed rules `table1`

In the second table, ARP spoofing, DHCP server spoofing, and IP(v6) spoofing are considered. The rule with highest priority (in Line 1 of Listing 3) only allows ARP requests and responses with non-spoofed addresses (Req. 4). These frames are also (like the ARP rule in **table0**) sent to **table2** which does ARP optimization. Rules number two and seven prevent DHCP server messages from VMs by dropping all packets that are sent from the DHCP server UDP port [8], [9]. This prevents the machines from faking a DHCP server (Req. 9). All other IPv4 packets are allowed by action `normal` in the next rule if the source address matches the configured one (Req. 2). The next three flows handle ICMPv6 spoofing. It is only allowed to send ICMP messages from one of its own IPv6 addresses (Req. 6). Additionally, the unspecified address needs to be allowed to permit Duplicate Address Detection. All these ICMPv6 messages are sent to **table3** to handle more ICMP security issues. Rules eight and nine allow all other IPv6 packets that leave the virtual machine with the correct link local or global IPv6 address (Req. 3).

```
1  in_port=* arp arp_op=1 arp_tpa=$ip4
       ↪priority=40 action=$port
2  in_port=1 arp arp_op=1 priority=2 action=drop
3  in_port=* arp arp_op=1 priority=1 action=1
4  in_port=* priority=0 action=normal
```

Listing 4: Installed rules `table2`

Now the rules of the ARP optimization table (**table2**) are explained. The first rule within Listing 4 defines that all ARP requests that are destined for any of the virtual machines are not broadcasted like ARP is usually done, but they are only sent directly to the corresponding machine (here no `in_port` is considered). Such a rule is created for every VM because of the VM's IP address within it. ARP request which come from the interface `upstream`, i.e., the Internet and were not matched against the first rule are dropped as these are destined to IP addresses which are not present within this network. All other ARP requests coming from one of the VMs are sent to `upstream`. All other ARP packets are processed the normal way (by defining the table miss with action `normal`). With this table Req. 8 is met.

```
1  icmp_type=134 priority=40 action=drop
2  icmp_type=136 nd_target=$ip6 priority=39
       ↪action=normal
3  icmp_type=136 nd_target=$eui64 priority=38
       ↪action=normal
4  icmp_type=136 priority=37 action=drop
5  icmp_type=137 priority=36 action=drop
6  in_port=* priority=0 action=normal
```

Listing 5: Installed rules `table3`

The fourth table represented in Listing 5 handles spoofing within ICMPv6. First, all router advertisements (ICMP type 134) sent from the VMs are dropped (Req. 5). This prevents the machines from faking to be a router to others in the network. The next three rules only allow sending Neighbor Advertisement messages from their own IPv6 addresses. All other Neighbor Advertisements are dropped (Req. 6). Additionally, all ICMPv6 Redirect messages are forbidden as this may lead to DoS attacks. With this rule, Req. 7 is also fulfilled.
In the qemu hooks file, the shutdown of the machines is also handled. The deletion of the rules is handled during step *stopped* and phase *end*. Here, all rules affecting the VM are deleted. This is done by matching the cookie of the flows to the MAC address of the VM.

5. Conclusion and Future Work

With the outcome of this paper, the network connecting the VMs is more secure. The implementation handles MAC, IPv4, IPv6, APR, NDP, and DHCP spoofing. Furthermore, the optimization of ARP requests aims for reducing the CPU load of the host system.
As the number of different layer two network attacks grows and new attacks are created over time, more security features need to be added in future. Currently, the problem of tiny IPv6 fragments is not addressed yet [10]. Additionally, the performance of checking the OpenFlow rules can be evaluated in future. An interesting point is whether the checks can be more performant if other strategies (blacklisting, whiteliting) are used. As multicasts and broadcasts can lead to a network overload, a meaningful prevention of this should be explored. A first idea would be rate limiting of the VMs when sending to much traffic into the network. Whether this can be realised is part of future work.

References

[1] S. Whalen, *An Introduction to ARP Spoofing*. Chocobospore, 2001.

[2] D. Plummer, "An Ethernet Address Resolution Protocol or Converting Network Protocol Address to 48.bit Ethernet Address for Transmissing on Ethernet Hardware," RFC 826, 1982.

[3] T. Narten, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, 2007.

[4] A. Pilihanto, *A Complete Guide on IPv6 Attack and Defense*. SANS Institute, 2011. Available at https://www.sans.org/reading-room/whitepapers/detection/complete-guide-ipv6-attack-defense-33904.

[5] S. Thomson, "IPv6 Stateless Address Autoconfiguration," RFC 4862, 2007.

[6] Y. Bhaiji, *Understanding, Preventing, and Defending Against Layer 2 Attacks*. Cisco, 2007. Available at https://www.cisco.com/c/dam/global/en_ae/assets/exposaudi2009/assets/docs/layer2-attacks-and-mitigation-t.pdf.

[7] Open Networking Foundation, *OpenFlow Switch Specification*, 2012. Available at https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf.

[8] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, 1997.

[9] T. Mrugalski, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 8415, 2018.

[10] A. Atlasis, *Attacking IPv6 Implementation Using Fragmentation*. Center for Strategic Cyberspace + Security Science, 2012. Available at https://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf.

Networking in MirageOS

Fabian Bonk, Paul Emmerich*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: fabian.bonk@tum.de, emmericp@net.in.tum.de

***Abstract*—MirageOS is a modern library operating system written in the functional, memory-safe OCaml programming language. Users of MirageOS write application code in OCaml and link against various libraries provided by MirageOS. These include a complete network stack (Ethernet, IP, TCP, UDP, TLS) written in pure OCaml as well as a number of backends for receiving and transmitting packets. We introduce some of MirageOS' techniques for handling raw memory. We detail two of the various networking backends offered by MirageOS as well as a library used for safe abstraction over raw memory. We additionally suggest possible performance improvements in MirageOS. Finally we compare MirageOS with *ixy.ml*, a small userspace driver for *ixgbe*-compatible NICs written entirely in OCaml.**

***Index Terms*—library operating system, unikernel, OCaml, networking**

1. Introduction

MirageOS introduces the concept of unikernels: Application-specific, standalone, bootable virtual machine images designed to run on top of a hypervisor (initially Xen, nowadays also KVM) [1]. The hypervisor provides hardware abstractions and isolation between unikernels. Unikernels are configured at compile-time to target a specific hypervisor and only include code to support their exact required features. Unlike the typical VM deployment that runs few services on top of an entire operating system such as Linux (including Linux's filesystems, drivers, network stack, userspace, etc.), a unikernel only includes the code it requires, e.g. a static webserver includes only a network stack and its hardcoded webpages. Anything that isn't strictly required is not included in the final unikernel, which leads to typical image sizes of a few MiB.

Figure 1 compares a typical VM deployment and a unikernel deployment.

MirageOS provides these libraries for many backends including a standard UNIX backend that runs the unikernel as a normal process, a Xen backend, and a KVM backend (via Solo5). These backends all provide specific implementations for MirageOS' interfaces, including MirageOS' `Mirage_net.S` network interface. In the following we analyse the KVM and Xen implementations of the `Mirage_net.S` interface.

MirageOS is written in OCaml[1], a multi-paradigm programming language that supports functional, imperative and object-oriented programming styles. OCaml features

1. https://ocaml.org/

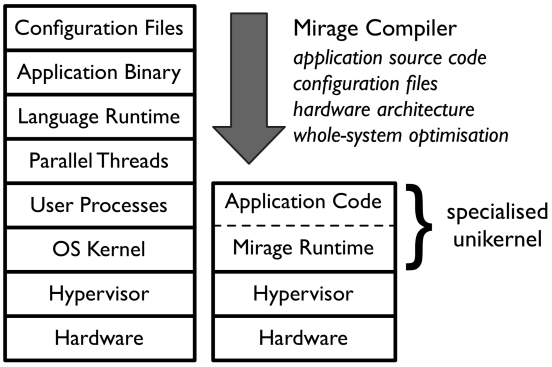

Figure 1: Typical VM deployment (left) vs. Unikernel deployment (right) [1]

```
module Main
  (N : Mirage_net_lwt.S)
  (C : Mirage_clock_lwt.PCLOCK) = struct
  let start n c =
    N.listen n (fun _ ->
        let now, _ = C.now_d_ps c in
        Logs.info
          (fun f ->
              f "got a packet at %d s!" now);
        Lwt.return_unit)
end
```

Figure 2: Example unikernel

memory safety, static type checking with type inference, garbage collection, and an optimizing native code compiler with support for multiple architectures. *Real World OCaml* [2] provides a good introduction to OCaml.

MirageOS users write their applications as OCaml functors. A functor is an OCaml module that is parameterized over other OCaml modules. In MirageOS' case the parameters are modules that implement functionality that is commonly provided by operating systems (network stack, file sytems, clocks, etc.).

Figure 2 shows a unikernel that simply prints a log message whenever it receives a packet. It is parameterized over a network interface (`N`) and a POSIX clock (`C`). At compile-time specific implementations for these modules must be chosen. The available choices depend on the runtime environment, i.e. which hypervisor or host OS will be used.

Section 2 introduces MirageOS' memory handling libraries. Section 3 explains MirageOS' network interface abstractions as well as two specific implementations of

doi: 10.2313/NET-2019-06-1_10

MirageOS network interfaces. Section 4 suggests some candidate performance improvements for MirageOS' network interface implementations. Finally section 5 compares MirageOS' network interface with ixy.ml, a network driver written in OCaml.

2. MirageOS Memory Handling

MirageOS needs to communicate with hypervisors using shared memory. Since OCaml natively only has limited support for accessing raw memory, MirageOS provides two libraries to handle page-aligned allocation and access in a safe way.

2.1. io-page

MirageOS provides *io-page* [4], a library for allocating page-aligned memory. io-page supports both UNIX and Xen backends (as well as Windows, though MirageOS itself doesn't support Windows). On Xen it uses Mini-OS' [5] `_xmalloc()` memory allocator. Mini-OS is a small kernel developed by the Xen project. MirageOS uses parts of it for CPU initialization, console output and memory allocation.[2] On other platforms (besides Windows) `posix_memalign()` is used for allocation.

2.2. cstruct

MirageOS uses a small wrapper library around C-like structures called *cstruct* [3] to facilitate safe and easy access to raw memory blocks. This library is split into the core cstruct library that manages the raw memory as well as a preprocessor called *ppx_cstruct* and the UNIX-specific library *cstruct-unix*.

2.2.1. cstruct. The main cstruct library defines an OCaml type `Cstruct.t` (referred to as cstruct from now on) that stores a reference to an OCaml Bigarray (which in turn references a raw memory region) as well as the array's length and an optional offset into the array.

The library includes functions for reading from and writing to these arrays in both little and big endian modes. Additionally cstructs can be converted to various other OCaml types such as `string`, `bytes` and S-expressions. Reads and writes are bounds-checked at runtime to ensure safety.

2.2.2. ppx_cstruct. ppx_cstruct is an OCaml ppx preprocessor that automatically generates accessor functions from C-like struct definitions (akin to LuaJIT's `ffi.cdef`).

Programmers simply declare the fields and types of `struct` and `ppx_cstruct` generates a number of functions for reading and writing each field as well as functions to hexdump an instance of the `struct`. Figures 3 and 4 show a UDP header definition and the values generated by ppx_cstruct respectively.

Additionally C-like `enums` can also be declared.

2. https://mirage.io/blog/introducing-xen-minios-arm

```
[%%cstruct
  type udp_header = {
    sport : uint16;
    dport : uint16;
    length : uint16;
    checksum : uint16
  } [@@big_endian]
]
```

Figure 3: cstruct UDP header declaration

```
val sizeof_udp_header : int
val get_udp_header_sport :
  Cstruct.t -> int
val set_udp_header_sport :
  Cstruct.t -> int -> unit
val get_udp_header_dport :
  Cstruct.t -> int
val set_udp_header_dport :
  Cstruct.t -> int -> unit
val get_udp_header_length :
  Cstruct.t -> int
val set_udp_header_length :
  Cstruct.t -> int -> unit
val get_udp_header_checksum :
  Cstruct.t -> int
val set_udp_header_checksum :
  Cstruct.t -> int -> unit
val hexdump_udp_header_to_buffer :
  Buffer.t -> Cstruct.t -> unit
val hexdump_udp_header :
  Cstruct.t -> unit
```

Figure 4: Values generated by `ppx_cstruct` from declaration in Figure 3

2.2.3. unix-cstruct. unix-cstruct wraps the `mmap(2)` system call and creates a cstruct by memory-mapping a file descriptor. The file descriptor is not mapped as shared (`MAP_SHARED`) but as private (`MAP_PRIVATE`), therefore writes to the cstruct are not reflected in the underlying file.

3. MirageOS Network Interfaces

The `Mirage_net` [6] module defines the module signature (interface) MirageOS programs use to send and receive packets. At compile-time a specific implementation that fulfills this signature must be chosen and linked into the unikernel.

While `Mirage_net.S` is an abstraction over network devices it itself leaves some implementation details abstract. All MirageOS backends actually implement `Mirage_net_lwt.S` which uses the Lwt library [7] for concurrency.

There are a number of different backends that implement this signature:

- mirage-net-unix [8]
- mirage-net-xen [9]
- mirage-net-macosx [10]
- mirage-net-flow [11]
- mirage-net-fd [12]
- mirage-net-solo5 [13]

doi: 10.2313/NET-2019-06-1_10

Sections 3.2 and 3.3 detail the hypervisor backends *mirage-net-xen* and *mirage-net-solo5* respectively.

MirageOS' network stack (layer 2 and up) simply calls into the network backend to communicate; the same network stack can be run on any backend.

3.1. `Mirage_net.S`

A MirageOS network interface must support these core functions:

- `write` transmits a single packet
- `writev` transmits a list of buffers concatenated into a single packet; this is generally implemented by concatenating the buffers into a freshly allocated, larger buffer and then transmitting this buffer
- `listen` calls a specified handler function for every received packet

The underlying implementation of the module signature may specify the type of packet buffers, asynchronous I/O operations, device state, MAC addresses and allocation operations for new buffers.

In the case of `Mirage_net_lwt.S` packet buffers are cstructs, I/O operations are Lwt promises and allocation is done using MirageOS' *io-page* library (see Section 2).

The `Mirage_net.S` signature also requires a number of other functions such as disconnecting from a network interface (interestingly connecting to an interface is not required), retrieving the interface's MAC address as well as reading the interface's receive and transmit statistics (bytes/packets sent/received).

3.2. Xen

The Xen implementation of the `Mirage_net_lwt.S` signature is written entirely in OCaml. It communicates with Xen via the netfront/netback protocol[3].

mirage-net-xen's `listen` loop sleeps until an event is fired on the event channel associated with the specified network interface. Once an event is fired a new cstruct is allocated for each received packet and the packet fragments delivered by Xen are assembled into the cstruct. MirageOS' handler function is called for every packet.

When transmitting, packet data is copied into a shared memory page and a reference to the page is stored in the transmit ring.

Both receiving and transmitting packets requires a full copy of the packet data from/to a shared page.

3.3. KVM

MirageOS' KVM implementation is provided by Solo5. Solo5 is an execution environment for unikernels. It can be used to run MirageOS unikernels on Linux's KVM hypervisor. On Linux it can interface with both *virtio* and TAP network interfaces, though virtio support is no longer maintained.

3. https://xenbits.xen.org/gitweb/?p=xen.git;a=blob;f=xen/include/public/io/netif.h

3.3.1. hvt. Solo5 provides a small hypervisor manager called *hvt* (hardware virtualized tender). hvt sets up a KVM virtual machine and runs a unikernel inside this machine. hvt connects to a TAP interface on the host operating system and forwards packets between unikernel and host.

3.3.2. mirage-solo5 and mirage-net-solo5. A unikernel interfaces with hvt via *mirage-net-solo5*, a small OCaml wrapper around *mirage-solo5* which in turn wraps hvt's hypercalls (hypervisor equivalent of a system call) and makes them callable from OCaml. OCaml cannot directly call C functions due to differing value representations. mirage-solo5 converts OCaml values to their C representation and vice versa.

mirage-net-solo5 implements MirageOS' `Mirage_net_lwt.S` signature.

The `listen` function repeatedly calls `solo5_net_read` function. If a packet has been received, it is written into a freshly allocated cstruct before MirageOS' handler function is applied to the buffer. If nothing has been received, the thread blocks until an I/O event is signaled by hvt. Note that the I/O event need not be a received packet; Solo5 currently offers no mechanism for waiting for specific I/O events.

mirage-net-solo5's `write` function calls `solo5_net_write`.

Both `solo5_net_read` and `solo5_net_write` simply call hvt's `hypercall_netread` and `hypercall_netwrite` which read from/write to hvt's TAP device.

4. Possible performance improvements

We identified some possible peroformance improvements for MirageOS' network interfaces.

4.1. Avoiding memory copies

MirageOS' Xen networking backend copies every single received and sent packet buffer to and from cstructs. Solo5 requires full copies to and from the host's kernelspace when sending and receiving packets respectively. Copying every packet's payload incurs performance penalties proportional to each packet's size, though it allows users to transmit any cstruct and keep any received cstruct forever. Additionally cstructs can be collected by OCaml's garbage collector; there is never any need for manual memory management.

Implementing "zero-copy" packet buffers will likely require modifications to MirageOS' APIs and its network stack.

4.2. Batching

MirageOS handles packets individually. Handling batches of packets could reduce per-packet overhead. Batching is a common pattern in high-performance networking toolkits such as the DPDK [14], Snabb [15] or ixy and its derivatives [16] (see Section 5). Implementing packet batching requires a modifications to MirageOS' `Mirage_net.S` signature and all of MirageOS' networking backends.

doi: 10.2313/NET-2019-06-1_10

4.3. Parallelism

MirageOS unikernels can only be run on a single CPU core at once. This limitation is imposed by OCaml's runtime. Once the *ocaml-multicore* [17] project reaches maturity, it may be possible for the Lwt library to upgrade its scheduler. Once Lwt supports parallelism it should require little effort to run MirageOS unikernels on multiple CPU cores, given that MirageOS' interfaces already support concurrency.

5. Comparison with ixy.ml

ixy.ml [18] is a userspace driver for *ixgbe*-compatible NICs (Intel 82599) written entirely in OCaml and targeting Linux machines. ixy.ml also makes use of cstruct.

5.1. Memory

ixy.ml does not use OCaml `lists` but rather stores all data elements in arrays. Arrays in OCaml are fast to traverse, and are mutable (i.e. can be modified in-place). Mutable state requires careful programming to prevent race conditions. Functional programming languages generally favor immutable data structures.

5.2. Packet buffers

ixy.ml uses Linux's *hugetlbfs* for memory allocation. It provides the `Ixy.Memory` module that allows users to create fixed size memory pools from which packet buffers can be allocated. These packet buffers contain cstructs that wrap part of a huge page (2 MiB page). A user accesses packet data directly in the hugepage through the cstruct library. This memory is ready for DMA (Direct Memory Access) and can immediately be read and written by the NIC.

ixy.ml requires explicit allocation and deallocation of packet buffers by the user due to the fact that packet data must be written to DMA memory. Packet data is never copied between buffers. Use-after-free cannot be detected, though should be avoided.

mirage-net-xen requires copying of packet data for both receive and transmit; users never directly write to the buffer read by Xen's backend driver and vice versa. mirage-net-solo5 triggers a copy of packet data between userspace and kernelspace when calling `read`/`write` on the TAP device's file descriptor.

Since packet data is always copied in MirageOS, the user may hold on to previously sent or received buffers. Therefore MirageOS provides more memory-safety guarantees than ixy.ml.

5.3. Receive/Transmit

All `Mirage_net.S` implementations only transmit and receive packets one at a time. ixy.ml implements batching; multiple packets are sent/received at once.

5.4. API

MirageOS generally implements I/O asynchronously. Most function calls that may block can be run in the background.

ixy.ml's receive and transmit functions do block though they never wait (unless explicitly told to and the NIC cannot keep up with the user program's transmit speed). Given that the NIC operates asynchronously there is never any need to wait. If there is not enough room in ixy.ml's transmit queue(s), any unsent packets are simply returned to the user program.

MirageOS' network functions don't require any manual memory management by the user. Any cstruct can be sent as a packet (assuming its size is within the MTU).

Figure 5 shows an implementation of a bidirectional layer 2 forwarder using ixy.ml's API. Initialization code has been omitted. See `app/fwd.ml`[4] in ixy.ml's repository for a full implementation.

Figure 6 shows an implementation of a bidirectional layer 2 forwarder using MirageOS' API. Initialization code has been omitted and errors will be ignored.

```
let forward rx_dev tx_dev =
  (* receive a batch of packets *)
  let rx = Ixy.rx_batch rx_dev 0 in
  (* transmit all packets *)
  Ixy.tx_batch_busy_wait tx_dev 0 rx

let () =
  let a, b = init_devs () in
  while true do
    forward a b;
    forward b a
  done
```

Figure 5: ixy.ml layer 2 forwarder

```
open Lwt.Infix

module Main
  (N_a : Mirage_net_lwt.S)
  (N_b : Mirage_net_lwt.S) = struct
  let start net_a net_b =
    Lwt.join
      [ N_a.listen
          net_a
          (fun frame ->
            N_b.write net_b frame >|= ignore)
        >|= ignore;
        N_b.listen
          net_b
          (fun frame ->
            N_a.write net_a frame >|= ignore)
        >|= ignore;
      ]
end
```

Figure 6: MirageOS layer 2 forwarder

4. https://github.com/ixy-languages/ixy.ml/blob/master/app/fwd.ml

doi: 10.2313/NET-2019-06-1_10

References

[1] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, “Unikernels: Library Operating Systems for the Cloud,” SIGPLAN Notices, vol. 48, pp. 461-472, March 2013

[2] Y. Minsky, A. Madhavapeddy, J. Hickey, “Real World OCaml,”, https://v1.realworldocaml.org/, 2013

[3] MirageOS project, “ocaml-cstruct,” https://github.com/mirage/ocaml-cstruct, 2019

[4] MirageOS project, “io-page,” https://github.com/mirage/io-page, 2019

[5] Xen Project, “Mini-OS,” https://wiki.xen.org/wiki/Mini-OS, 2019

[6] MirageOS project, “mirage-net,”, https://github.com/mirage/mirage-net, 2019

[7] Ocsigen Project, “Lwt,” https://ocsigen.org/lwt/4.1.0/manual/manual, 2019

[8] MirageOS project, “mirage-net-unix,”, https://github.com/mirage/mirage-net-unix, 2019

[9] MirageOS project, “mirage-net-xen,”, https://github.com/mirage/mirage-net-xen, 2019

[10] MirageOS project, “mirage-net-macosx,”, https://github.com/mirage/mirage-net-macosx, 2019

[11] MirageOS project, “mirage-net-flow,”, https://github.com/mirage/mirage-net-flow, 2019

[12] MirageOS project, “mirage-net-fd,”, https://github.com/mirage/mirage-net-fd, 2019

[13] MirageOS project, “mirage-net-solo5,”, https://github.com/mirage/mirage-net-solo5, 2019

[14] Linux Foundation, “Data Plane Development Kit,” https://dpdk.org/, 2013

[15] Luke Gorrie et al., “Snabb: Simple and fast packet networking,” https://github.com/snabbco/snabb, 2012

[16] Paul Emmerich et al., “ixy-languages”, https://github.com/ixy-languages/ixy-languages, 2018

[17] OCaml Labs, “Multicore OCaml,” http://ocamllabs.io/doc/multicore.html, 2017

[18] Fabian Bonk, “ixy.ml,” https://github.com/ixy-languages/ixy.ml, 2018

doi: 10.2313/NET-2019-06-1_10

Bot-based IT Troubleshooting

Benjamin Braun, Jonas Jelten*, Simon Bauer*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga82vah@mytum.de, jelten@net.in.tum.de, bauersi@net.in.tum.de

***Abstract*—The use of relatively newer technologies like chatbots and artificial intelligence has seen a steady rise for customer service encounters in various companies, in order to minimize cost and personnel while providing a broad access point with an individual feel to it to the general customer base. The state of the art of these chatbots and their use cases is compared for a general overview, as well as their applicability to customer service in IT system administrations and service providers. The difficulties and problems of this specific application, in comparison to other uses of chatbots, are highlighted and discussed.**

***Index Terms*—chatbots, troubleshooting, customer service, service encounters, information technology**

1. Introduction

The improvement of customer services in efficiency and effectiveness has always been an ongoing issue across many fields. One of the most promising approaches to solve this challenge is the use of technologies such as artificial intelligence or chatbots, which have been developed heavily over the last decades and seen some significant advancements in recent years.

Chatbots are programs working as a dialog system for human-computer interaction, which emulate human behaviour in a conversation and can provide both chatter and serious help with tasks and questions of the user.

The goal for customer services is to enhance the user experience in service encounters by providing immediate and exact responses, which are available at all times, something that is difficult to achieve and costly with customer service personnel. Companies might have a great interest in reducing the human factor in these matters, for both cost and efficiency, but at the same time the experience with customer service should not become more impersonal.

It is the responsibility of the service providers to solve the customers' problems by not only making the desired knowledge available for look-up, but also offering creative and custom solutions to the individual issues.

To bring together these desired aspects of customer service, chatbots seem like one of the most ideal approaches, which is proven by the already manifold uses across many different websites of companies.

2. Value of Chatbots

An ideal chatbot would behave indistinguishable from a real human working in customer service, but would also be much faster, always available and with perfect knowledge and memory. While there is no perfectly human-like or sentient artificial intelligence yet, this concept of the ideal customer service helps outlining some of the most important values and benefits of chatbots, and serves as the long-term goals for these kinds of technologies.

2.1. Advantages over conventional customer services

The most important aspect of having a chatbot, in place of, for example, just a FAQ-section on their website, is the personal nature of this access point to the knowledge databases. Customers can talk to them about their problem or ask questions, and then it is within the responsibility of the program to process the query and provide the corresponding answers. The workload of searching for relevant information is taken away from the customers. In addition, if further information is needed, the program can selectively ask for additional data.

This possibility of a dialog about the specific issues of the user is the usual reason for consulting customer services anyway, and it is of great importance that chatbots keep this nature of interaction preserved.

Some of the more obvious advantages that chatbots provide, in opposition to human customer services, are speed and availability. In order for service personnel to compete with the speed of a chatbots' direct link to their databases, they would need to be experts on the corresponding field and memorize the solutions to most issues.

Even if such experts could be hired for customer services, it would need an unfeasible amount of employees to cover the availability that chatbots offer. The software is available 24/7 and no matter how many clients are already in dialog with the customer service, a new request should never be placed on hold, as long as there is enough server capacity to run and process all queries.

2.2. Benefits in costs

Especially the before mentioned last two advantages come hand in hand with another significant benefit. The cost of a bot-based customer service is overall much lower than employing humans. On top of running costs, chatbots do have a substantial initial implementation or setup price though, which depends on the method used to acquire the bot, as well as the extend of its functionality.

There are several ways of obtaining a chatbot as discussed by Viktoria K. and Vlad V. in [1]: Buying an

doi: 10.2313/NET-2019-06-1_11

already existing solution, using self-service platforms to build and design chatbots, or implementation from scratch. After the initial setup costs, there might also be maintenance costs, for analysis, adjustments and improvements, or further bot training if needed.

While developing and integrating a chatbot can cost from a few thousands to several tens of thousands of dollars (Estimations by [1], [2] and [3]), the low running costs make up for it very quickly; and the comparison to the unrealistic customer service team which would be needed to even come close to a chatbots' knowledge, speed and availability, makes the advantage in costs very apparent.

These comparisons are disregarding one important fact though: chatbots are not yet competent enough to cover all customer service needs for any company. The usual application of bots entails taking care of a majority of queries from customers that are repetitive or easy to look up. For the more complex user requests that are to difficult to understand for the software, the bot usually refers the client to human staff. Therefore it is not yet possible to dispense with all employees, but their workload can be greatly reduced.

Despite the requirement of some form of remainder human staff alongside chatbots, the advantages of using such technology in customer service still hold and the subsequent reduction in costs is still significant.

An estimating study by Isabella Steele in [4] shows that even if only 50% of queries are eligible for chatbots (this percentage is predicted to go up to 90% in the next years), the deployment of a bot can cut the costs on staff by at least 44%.

As a side effect, it is also proposed in [4] that the shift in tasks for human employees, away from repetitive and dull questions and towards mostly complex and skill-challenging queries, leads to a more fulfilling experience for the service agents, and happier, more motivated staff leads to more satisfied customers.

Beyond all extrapolations of return values, chatbots are a long-term investment for any company. The requests for chat-based customer support saw a rise of 180% from 2016 to 2017 as stated by [4]. The number of queries a bot can easily answer goes up in a similar fashion; not only because of the higher amount of requests, but also as a result of the rapidly evolving technologies behind these bots, the complexity of queries artificial intelligence systems are able to handle is increasing quickly.

In conclusion, chatbots are already very profitable and capable, and they will continue to grow in both factors.

3. Current Use Cases

The advantages of using chatbots or similar technologies has already been realized by many companies across many fields of use, although there is still much room for improvement.

Chatbots have been researched and developed since the 1960s; Joseph Weizenbaum's ELIZA, published in 1966, is believed to be one of the earliest working bots. It was designed to act like a Rogerian therapist, but its main functionality consisted of using the recognition of keywords and the output of corresponding answers to fool people into the assumption that the machine was actually intelligent [5].

This trend shaped the development of chatbots for many years, the research was mostly aimed at passing the Turing Test rather than commercial use. Especially the Loebner Prize Competition, which is the first formal instantiation of the Turing Test and has been giving out prizes for the most human-like chatbot annually since 1991, has sparked some controversies, "whether this competition is really contributing to the development of AI, or it is blocking it", as stated by Bradesko et al. in [6].

Nonetheless chatbots designed for actual commercial tasks have emerged in many different domains, with varying functionality and applicability.

A study conducted by Prof. Dr. Julian Kawohl and Stefanie Haß in [7] investigated the use of chatbots by DAX and MDAX corporations. Their findings conclude that 15% out of the 80 examined companies have bots available for their customer experience. Whether this amount is too low for the present times and proof of the slow adaptation of big, established corporations, or a sign of progress and proof of the usefulness of bots in various fields, is debatable.

At the very least the twelve inspected chatbots of these big companies give an insight into the diversity of commercial applications. Some of these chatbots are only designed for a very specific purpose, showing that most of even these few companies are still far from an AI supported customer service. Some examples for the chatbots' purpose in life, as listed by [7], are:

- Booking of sport courses at Adidas Gym London (Adidas)
- Calculation of automobile insurance (Allianz)
- Broadcasting tool for touring car tournaments (BMW)
- Assistant for spot removal (Persil)
- Information as to flight prices (Lufthansa)
- Assistant for searching for second-hand cars (VW)
- News service, primarily for German Football League (Bild)
- Personal shopping consultant (Zalando)

Other chatbots are listed with the broader term "personal customer consultant", implying a more general field of application across the companies' offers.

The very varied and in some cases extremely specific uses of chatbot technology show that at least big German companies are slowly exploring the market with the introduction of artificial dialog systems, but are still far from the potential of this innovative medium.

3.1. Comparison of the capabilities of different chatbots

While the functionalities of different chatbots overlap for the most parts, roughly three roles can be determined for commercial bots: personal assistants, shopping assistants and troubleshooting.

3.1.1. Personal assistants. The most well known types of chatbots are the personal assistants, due to just about every big IT company building their own virtual assistant following the success of Apple's "Siri" in 2011. Amazon

doi: 10.2313/NET-2019-06-1_11

has "Alexa", Google offers their "Google Assistant", Microsoft followed with "Cortana", Samsung has "Bixby", and the list continues.

Although there is not quite a consensus yet whether there is a difference between virtual assistants and chatbots, or one of these words is just an umbrella term for the other [8], they share enough significant features to be worth mentioning.

The basic services of these assistants include providing current information and facts, setting alarms or timers, adding or retrieving events from the calendar, calling or texting specific contacts, accessing media libraries or streaming services, and many more.

This shows their early intended use as a voice commanding tool for the phone, while also having the gimmick of chitchat. The ability to talk with the user and to tell jokes etc. was something new and exciting at the time and surely helped their popularity.

As times and bots made progress over the years, they also gained some more significant capabilities. Especially "Alexa" and "Google Assistant" acquire more and more service features, ranging from controlling various aspects of smart homes to buying products online with just voice commands [9].

Recently Google pushed another advancement for self-dependence of chatbots, the ability to let the virtual assistant call services like shops or restaurants and scheduling appointments or asking for information, as presented recently in the form of "Google Duplex", see [10]. While this concept has yet to become reality and widely available, this great leap in technological advancement shows that virtual assistants will continue to grow more potent.

3.1.2. Shopping assistants and troubleshooting bots. The other two roles of chatbots are much more similar to each other in opposition to personal assistants, and also tie in better with the issues discussed in previous sections. Both shopping assistants and troubleshooting bots are chatbots for customer service, and quite often the systems are capable of fulfilling both roles.

As, for example, the before mentioned study in [7] showed, being a pure shopping/searching assistant is still the more common task of bots, primarily because they get used by companies that want to improve their sales and do not have a very consistent troubleshooting process, for example clothing shops or travel agencies.

Due to the large amount of small and very specialized chatbots in this domain, it is hard to get a generalized overview of their capabilities, but some of the usual details can be summarized.

- Chatbots tend to present themselves as a visible dialog partner, with the help of avatars or similar, and can be written with some form of personality to reinforce humanness and sympathy.
- The basic framework of a well-constructed chatbot contains common phrases for small talk and chatter, and some general knowledge.
- The chatbot needs to be capable of understanding and outputting natural language. Some bots only use keyword searches to guess the query, others are able to process full sentences, but specific technical inputs by the user should never be required.
- Shared details about the client are often stored and retrieved when necessary to personalize the dialog.
- Bots have access to a database of the relevant knowledge, in order to answer questions, but also to proactively show the user additional content.
- Whenever the system realizes it can not sufficiently help the client anymore, it refers them to supplementary services, usually human staff.
- And in general, chatbots are available for free around the clock, and have the ability to answer queries in reasonably short time, with high enough effectiveness and efficiency to satisfy the customer.

These general qualities for chatbot systems in customer support are referring to the metrics used in the study by Kawohl and Haß in [7], which most of the tested chatbots satisfied to an acceptable degree.

Additionally, chatbots also used as troubleshooting customer service are able to ask the user in a search tree-like fashion about several "symptoms", until the most likely problem is found and a solution can be proposed. For this role the systems have to be capable of identifying the best route for questioning the client about their problem with very little starting information given. An elaborated tree structure for all possible problems and how to identify them is crucial for this kind of customer service.

Two examples of commercial service bots will be examined in more detail.

At first a rather negative exemplar: Ikea's chatbot "Anna" was one of the earliest better known customer service bots in the industry, and was online for over ten years, before being retired with no plans of replacement in 2016. "Anna" aimed at answering questions and guiding customers around the website in an interactive way, but users seemed to be too frustrated with the bot. The developers tried too hard to be natural and human-like, and diverted too far from the real purpose of the bot, to provide correct answers as efficient as possible. "Anna was too human", according to an Ikea representative [11].

This case is especially interesting because the usual complaints about chatbots deemed them "overly robotic and lacking a personal touch" [11], signifying the importance of balance.

The second example is an application of a chatbot, which is not from a specific company, but rather available as an add-on extension for browsers. The bot called "SuperAgent" leverages publicly available e-commerce data in the form of product descriptions and user generated data like Q&As and reviews, to generate a customer service dialog in-page of online shopping websites like Amazon or Ebay, as presented by Cui et al. in [12].

It uses state of the art natural language processing and machine learning techniques to obtain the desired information from the available data. This innovative technology has been shown to improve end-to-end user experience for online shopping and to make large amounts of information and user generated content easier comprehensible [12].

"SuperAgent" demonstrates effectively how the pinnacle of customer service bots in the form of shopping assistants is not yet reached, and that there are more ways to enhance the customer experience besides just the

doi: 10.2313/NET-2019-06-1_11

hope for every company to provide a decent and capable chatbot.

3.2. Application in IT

Information Technology (IT) companies are not utilizing chatbots as much yet, which is surprising, considering these systems are developed in this industry sector.

The website "chatbots.org" maintains a list of registered chatbots, virtual assistants and conversational agents all over the world, and has them categorized by countries, languages, platforms and consumer themes. Their archive contains 1368 chatbots as of December 2018.

And yet, an extensive search through chatbots in the category "electronics and hardware" and "telecoms and utilities" yielded in just about one accessible and visibly troubleshooting chatbot from an IT company. This bot is Dell's "Assisted Search" (see [13]), and it is built for troubleshooting common technical issues with dell products. This system is not much of a conversationalist, and rather quick to direct towards articles with solutions to the identified problems. After the initial typed-in question, the user is mostly navigating with links provided by the bot to further specify the issue, until solutions from the article database are presented, or the agent knows some short tips, which are given directly in chat.

In contrast to the rather plain "Assisted Search", T-Mobile's "Tinka" has a lot of personality and even a fictitious backstory (see [14]). This chatbot is an assistant for products and the website, but also offers customer support regarding accounts and contracts, as well as tech issues. Some basic functionalities of troubleshooting are therefore integrated in this bot, as it is the case with many customer service chatbots.

Finding only very few cases of troubleshooting bots in IT does not mean there are no other, the list from "chatbots.org" is by no means complete. But it does show how underutilized this technology is in the context of IT, a domain with a lot of requests for troubleshooting.

4. Difficulties and Problems in the IT Domain

As Subsection 3.2 showed, the use of chatbots in IT troubleshooting processes is still very much improvable. But as discussed throughout Subsection 3.1, the qualities of troubleshooting bots are not that different from other customer service systems, so wherein lie the difficulties?

Not much information can be found on this topic, therefore this section will try to discuss possible issues and reasons for this discrepancy.

Troubleshooting in IT services is done by defining the problem, analyzing possible causes and following them to the root, and then looking for the best fix.

One of the most important difficulties with this approach are often the users themselves: technological understanding can occasionally be extremely limited. "It does not work" can be a very challenging starting point for any customer service to try and fix the issue, human or bot alike. Guiding such a client through the process of finding possible causes can become even more of a sticking point.

Chatbots would need to have a way of identifying the user's affinity and knowledge about technology, and adapt accordingly. The search tree for analyzing the symptoms and causes needs to be both very general and elaborate, but also detailed and specialized enough to actually identify the correct issues. Users with greater technological understanding would also want the bot to respond on a higher level.

This makes the setup of databases and search trees quite difficult and much more extensive than with other types of chatbots.

A sophisticated natural language processing system is necessary for the same reasons, it has to be adaptable in its level of communication and recognize how much technical jargon it can and should use with the client.

On top of that, the language processing system needs to be able to extract information from even the most vague descriptions of the problems of customers, as well as identifying potentially wrong "self-diagnoses" and proposing some other possible causes for the issue proactively (Monitors have been wrongly blamed for enough computer problems already).

The extensiveness of possible issues in information technology, together with the unpredictability of the users and their knowledge, is most definitely a significant reason for the relatively unexplored potential of chatbot-supported troubleshooting services in IT. Although current chatbots are capable of fulfilling the role of general customer service, the additional competence needed to build an adaptable, more-than-decent chatbot for IT troubleshooting might be what is holding the bots' advancement in this industry back.

5. Conclusion and Future Work

Chatbots have some very distinctive advantages in the domain of customer services, they are always available, fast and have a big knowledge base. On top of that, they can reduce a company's cost for customer service by a significant amount while also increasing the work satisfaction for human employees.

Development over the recent decades assured that the capabilities of chatbots and virtual assistants are growing exponentially, and while they might not always be used to their full potential yet, companies are finding more and more ways to integrate dialog systems into their service experience.

As for the IT sector, it became apparent that the utilization has not proceeded as much as in other fields. Chatbots are still very much applicable in this domain and could bring great benefits for service providers and clients alike, but several additional difficulties demand for a more sophisticated and extensive creation of a chatbot.

In order to have an excellent troubleshooting bot for issues in IT, the varying amount of technical understandings of users and the extended amount of problems and possible causes need to be countered by extraordinary adaptability and a higher than usual intelligence of the assistant.

For the future, the next step would be to go into more detail about the technical possibilities of creating such a chatbot for IT customer service, investigating any current projects from the industry further, and working together towards the creation of a new chatbot system, built for the specific purpose to address the ongoing issues.

doi: 10.2313/NET-2019-06-1_11

References

[1] K. V. and V. V., "How Much Does it Cost to Build a Chatbot", Retrieved December 13th, 2018, from https://rubygarage.org/blog/how-much-does-it-cost-to-build-a-chatbot

[2] Borisov H., "Anatomy of a Chatbot - How Much Does it Cost to Build One?", Retrieved December 13th, 2018, from https://www.progress.com/blogs/anatomy-of-a-chatbot-how-much-does-it-cost-to-build-one

[3] Kaarma J., "How much does a chatbot cost?", Retrieved December 13th, 2018, from https://chatbotslife.com/how-much-does-a-chatbot-cost-783bf583ac4

[4] Steele I., "It's All About The $$$ - How Much Money Can Chatbots Actually Save You?", Retrieved December 13th, 2018, from https://www.comm100.com/blog/how-much-can-chatbots-actually-save-you.html

[5] McNeal M. L. and Newyear D., "Introducing chatbots in libraries", Library technology reports, vol. 49(8), 2013, pp. 5-10.

[6] Bradesko L. and Mladenic D., "A survey of chatbot systems through a Loebner prize competition", Proceedings of Slovenian Language Technologies Society Eighth Conference of Language Technologies, 2012, pp. 34-37.

[7] Kawohl J. M. and Haß S., "Customer Service 4.0 - Wie gut sind Chatbots?", from https://www.heise.de/downloads/18/2/5/4/1/3/4/2/Studie_chatbots.pdf

[8] Lobo J., "What is the difference between a chatbot and a virtual assistant?", Retrieved December 16th, 2018, from https://www.inbenta.com/en/blog/difference-chatbot-virtual-assistant/

[9] Martin T. and Priest D., "The complete list of Alexa commands so far", Retrieved December 16th, 2018, from https://www.cnet.com/how-to/amazon-echo-the-complete-list-of-alexa-commands/

[10] Leviathan Y., "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone", Retrieved December 16th, 2018, from https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html

[11] Brandtzaeg P. and Følstad A., "Chatbots: changing user needs and motivations", Interactions, vol. 25(5), 2018, pp. 38-43.

[12] Cui L., Huang S., Wei F., Tan C., Duan C., and Zhou M., "Superagent: a customer service chatbot for e-commerce websites", Proceedings of ACL 2017, System Demonstrations, 2017, pp. 97-102.

[13] http://www5.nohold.net/Dell/Loginr.aspx?pid=3&userrole=USen, Retrieved February 11th, 2019

[14] https://www.t-mobile.at/tinka/, Retrieved February 11th, 2019

doi: 10.2313/NET-2019-06-1_11

Client Monitoring with HTTPS

Felix Hartmond, Simon Bauer*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: felix.hartmond@tum.de, bauersi@net.in.tum.de

***Abstract*—With the increasing number of users and devices on the internet, it is more and more important for an administrator to know who and what is using his networks and services. Therefore it is necessary to find out information about the clients. This process is called fingerprinting. With the ongoing deployment of HTTPS information from the application layer can no longer be read directly. In this paper, we will look at the different layers of the HTTPS protocol stack and examine which fingerprinting approaches are still possible, or even only possible, with the added encryption.**

***Index Terms*—https, client fingerprinting**

1. Introduction

1.1. Motivation

With the growing number of users and devices on the internet is it more and more interesting who and what is producing traffic. Information about the users and devices is helpful for a lot of things. For example, it can be used to optimize services and networks. It can also be used to improve the security of a system as it can help to identify potential malicious actors. Such actors can also be tracked with such information which allows finding out more about them.

Getting information from network traffic is getting harder with the extended use of the encrypted version of the HTTP protocol: HTTPS. But there are far more ways to find out information about a client than just reading plain HTTP requests. And, even if encryption makes the analysis harder, it does not make it impossible. In this paper, we look at the different layers of the network stack and examine different ways to analyze a client with information provided by the different layers.

1.2. Outline

In Section 2, we will give a rough overview of different kinds of fingerprinting. After this, we will look at the involved protocols in an HTTPS communication and analyze which kinds of fingerprinting are possible on the different layers. We will go down the network stack from top to bottom and look at each layer. For each layer, we will analyze which methods were developed to gather information about the client from the information provided by this layer.

We will start at the very top and look at the Hyper Text Transfer Protocol (HTTP) in Section 3. Then, we look at the Transport Layer Security protocol (TLS), which is the new layer in HTTPS, compared to HTTP, in Section 4. After this, we examine in Section 5 the Transmission Control Protocol (TCP), which is usually used by HTTP connections as transport protocol [7] together with the Internet Protocol (IP). We look at these two layers at once as many approaches use information from both protocols. Finally, we will take a brief look at transport layer protocols. As these protocols are not used end to end they have some limitations we will also look at.

In the end, in Section 7, we will recap which information can be gathered overall.

2. Fingerprinting

Fingerprinting can be split into two categories: device fingerprinting and user fingerprinting. The methods used for fingerprinting can also be categorized into active methods and passive methods.

Device fingerprinting is about gathering information about the device which is communicating. These things include the operating system, drivers, protocol implementations or browsers. These can be very interesting to understand which device types are active in the network and which software in which versions they use. This information can be relevant to be able to optimize services and network infrastructure for the way how they are used. User fingerprinting is about getting information about the user who is actively using the applications. This can be interesting for tracking a user across different devices or services to be able to optimize, for example, the presented content for this user.

Fingerprinting methods can be split up into two categories: Active and passive methods. Active methods actively send probes to a device and analyze the responses. These methods require the ability to send probes to a device but are very mighty if there is no firewall in front of the device. They are mighty because due to the active participation of the observer it is possible to send specifically crafted packets to the client. On the other side, passive methods do not send out any new traffic, they just analyze traffic which is passing. Such methods completely rely on the traffic which is already there. But they has the huge advantage that they do not have to happen real time. It is possible to just capture and store traffic and do the complete analysis afterwards or even running an analysis on a traffic capture which was not stored with fingerprinting in mind.

doi: 10.2313/NET-2019-06-1_12

3. HTTP

If we would not use HTTPS, the information from the application layer would tell us a lot about the client and its action. There would be a lot of HTTP headers sent with every request, for example, the User Agent, which explicitly tells which system is used by the client. Additionally, we would be able to see all the requests including all parameters which gives us very detailed information about user and his actions. But, HTTPS hides all this application data through the addition of encryption. For the encryption an TLS layer is added between the TCP layer and the HTTP layer. This layer acts as an container for the HTTP layer and encrypts all data from it. So, theoretically, all data from the HTTP layer should not provide any information to an observer. But a lot of research was done to find out if it is possible to gain information about a client despite the presence of encryption.

Stöber et. al. [19] took a look at mobile applications and tried to find out which application observed traffic belongs to. They were able to identify the application with a success probability of 90% despite not seeing the content of the traffic. For they approach they utilized that a lot of application traffic is not directly triggered by a user action. Instead, most traffic is done by the application is the background on a reglar basic. They analyed the patterns of this background traffic and were able to deduce the running application from the traffic patterns.

Zion et. al. [13] tried to find out the client's operating system, the used browser and the application from encrypted traffic. They used supervised machine learning to assign labels of the form (OS, Browser, Application) to the packets. They were able to successfully classify packets with their approach. This shows that is is possible to get basic information about a user by only looking at the encrypted traffic.

Panchekno et. al. [14] analyzed if it is possible to find out which website was accessed if the traffic is protected and anonymized by anonymization networks like TOR. For their approach, they used a support vector machine (another machine learning technique) and were able to archive a true positive rate of 73% for a false positive rate of 0.05%.

Also, Cai et. al. [2] looked at encrypted traffic in the context of defenses like Tor. They used a simple model of network behavior to find out which homepage is accessed by a user and were able to identify the requested page with a good success rate.

Overall research has shown that encryption can not completely hide information from the communication. A very impressive example of getting information out of encrypted packets was presented by Wright et. all. They took a look at Voice over IP communication and were able to reconstruct spoken text from encrypted traffic. This was possible as the audio was encoded with a codec with variable frame rate which caused network packets of variable length. These varying packet lengths were sufficient to reconstruct spoken phrases. [22]

Dyer et. al. [4] took a conceptional look into the problem of hiding information completely through encryption or obfuscation. They came to the conclusion that it will always be possible to extract some kind of information from encrypted traffic as long as bandwidth optimizations are done, what every protocol does.

4. TLS

The Transport Layer Security Protocol takes care of encrypting all data above the TLS layer. To be able to do encryption the protocol has first to do a handshake between client and server. As there are no shared secrets between client and server initially, the first messages of the handshake are unencrypted. In the beginning, the client sends a so called "hello message" in which it tells the server about its version, cipher suites, compression methods and extension they support. The only other unencrypted message from the server contains the servers public key. After this, no more messages from the client are unencrypted. [3]

As often multiple homepages are hosted on a single server and these homepages use different certificates, TLS needs to know the requested homepage already during the handshake to provide the correct certificate. As ordinarily the domain is presented first after the finished handshake TLS has an extension called Server Name Indication (SNI). This extension presents the requested domain in the client hello message of the TLS handshake to make it possible for the server to present the correct certificate for the requested domain afterward. But the information from this is very limited as only the domain can be retrieved but neither any further information about the request nor details about the requesting client. [6]

To get more information about the client from the client hello message, Martin Husák et. al. went through the different values included in the client hello message and analyzed them towards the differences in the values for different client types. They found out that the most values are quite similar for different client types, but they found one very interesting value: the list of supported cipher suites. This list of support cipher suites differs enough between different clients to make it possible to distinguish them. [10]

Martin Husák et. al. used two methods to build a codebook which maps cipher suite lists to user agents. For the first one, a server based one, they logged the cipher suite lists as well as the user agent of incoming https connections on a web server. This method produces very accurate results but the clients have to visit a specific server to be included in the codebook. Because of this, they combined the fist with a second method which analyzed network traffic which includes HTTP as well as HTTPS connections from the same clients. By matching the connections from the same client together they were able to extend the codebook with clients which did not connect to their servers. In their tests, the top 10 cipher suite lists covered 68.5% of the network traffic and the top 31 cipher suite lists covered 90% of the traffic. Over their measurements, they discovered 1598 different cipher suite lists. [10]

Despite a very good variance in the cipher suite list there were still multiple user agents which correspond to a single cipher suite list. So they used a tool which extracts information about a system from a user agent like browser name, operating system or vendor. With this, they were

doi: 10.2313/NET-2019-06-1_12

able to classify the traffic into several categories for devices (desktop, mobile, unknown) as well as applications (browser, command line, application, update, unknown). [10]

5. TCP/IP

The Transport Control Protocol (TCP) takes care of the reliability network connection. It takes care of retransmission of lost packets, flow control, congestion control and multiplexing of connections between two hosts. [17]. The Internet Protocol (IP) provides addressing between the hosts which communicate with each other. Also, packet routing is done by this network layer. [16]. As many fingerprinting approaches use information from these two layers together, we will look at these two layers together.

The TCP specification, as well as the IP specification, does only describe the scenarios of regular operation. They do not specify corner cases like how an implementation should behave in the case of packets which should never occur, like strange combinations of flags or if TCP segments or IP fragments overlap. As these scenarios are not specified, different implementations have different behavior when processing such packets. This can be used for active fingerprinting. By sending such impossible packets the used implementation of the communicating host can be examined. For example, the tool NMAP implements a test where a packet with a FIN flag is sent without establishing a connection beforehand. Even if this packet should be just discarded, some implementations send an answer to it. [18]

In addition to implementation differences caused by edge cases and implementation bugs, TCP implementations also have slight differences in their behavior concerning retransmissions and congestion control. Different implementations have slightly different retransmission timeouts. By sending a TCP SYN packet, and measuring the time between the SYN-ACK packet and his retransmissions it is also possible to detect the used implementation as Veysset et. al. showed. [21]

Many more people have done research on active TCP/IP fingerprinting and developed tools which can execute the proposed measurements and classifications. Grek Taleck took a very detailed look at the different aspects of a TCP/IP implementations and developed a tool called SYNSCAN which's "objective is to fingerprint every aspect of a TCP/IP implementation". [20] Ofir Arkin and Fyodor Yarochkin developed Xprobe2 which uses an approach based on confidences to find a result, instead of relying on an exact match with a known fingerprint. [1]

All methods mentioned for TCP/IP so far were active methods. But an active method needs either an accessible device or a connection from this device to a controlled server which then can send probes as replies. Having an accessible device is rather easy for servers but in general, clients are hidden behind a firewall which makes it impossible to send probes to them. Passive fingerprinting methods allow fingerprinting just by analyzing the traffic from a client at any point on the path between client and server.

Different researchers created tools which analyze initial values of different fields from the TCP and the IP header. The siphone tool was a proof-of-concept tool which only analyzed the TCP window size, the IP Time to Live and the IP Don't Fragment bit from packets of a TCP connection. The p0f tool and the ettercap tool look especially at the SYN packet of the TCP connection. Ettercap additionally analyzes the SYN-ACK packet. In addition to the features of siphone, they also analyze different options and flags from the TCP header. [12] [23]

Vern Paxson took a different approach and developed a passive fingerprinting tool called tcpanaly. This tool analyzes a complete flow and looks at the congestion control behavior of the implementation as different implementations differ in their rounding of TCP's sstresh. It also takes a look at response delays for the creation of acknowledgments and watches for occurrences of a bug caused by uninitialized variables. [15]

TCP has a timestamp option which is intended for round trip time measurements. Kohno et. al. showed that it is possible to use the values from this timestamp option to detect clock skews which can be used to uniquely identify a user. Even if these variations are very small, they found out that it is possible to use that technique even if the observer is quite far away from the client. [11]

6. Link Layer Protocols

When looking at network layers below the IP layer, fingerprinting on these layers has a different initial situation - it is link dependent as first the IP Protocol takes care of the end-to-end addressing of a packet. So, when looking below the IP Layer we no longer look at packets, instead look at frames which only exist for one hop in the connection. This means, to be able to monitor a client we have to be very close, especially on the same link as the client. So fingerprinting on these layers is not possible for server administrators who want to know something about the clients sending requests. But for administrators who want to learn about the clients in their local network, these methods are useful.

Fingerprinting on the Link Layer is of course highly dependent on the used Link Layer protocol. When looking at an Ethernet frame the header only contains the source and destination MAC addresses, a type id of the content of the next higher layer and an optional VLAN tag. One of the few things which can be derived from these header fields is the vendor of the sender's network card as the MAC address contains an "Organization Unique Identifier" (OUI). [5] But the network card vendor does not give any reliable information about the software running on the client. Despite that, MAC addresses can be modified by a user or in case of virtual machines are not bound to physical hardware at all.

But, this situation is different for 801.11 networks. 802.11 is far more complicated than ethernet as it has connection establishment and authorization mechanisms, so it has far more potential to reveal information about the client.

Franklin et. al. used the active scanning for driver fingerprinting. Every 802.11 client periodically sends probe requests to discover access points in range. The time intervals between the probes can be used to identify the used driver. A huge advantage of this approach is that no specialized equipment is required for the measurements as

doi: 10.2313/NET-2019-06-1_12

the probes can be received any ordinary 802.11 hardware. [8]

It's even possible to get a step further and look at the analog signal created by an 802.11 interface. Gerdes et. al. took this approach and found out that a device can be identified and tracked by small differences caused by hardware and manufacturing inconsistencies. With this approach, it is not possible to identify the used driver or operating system, but it allows to identify a user on the link layer even if he changed his MAC address to another value or moves to another network. [9]

7. Conclusion

We have seen that even if the presence of encryption makes the analysis of the application layer harder it does not block fingerprinting. Despite the encryption, it is still possible to find out a lot about the actions of the user on the application layer. Especially with the current development in machine learning approaches, it is very hard to hide information from the application layer completely. The added TLS layer provides encryption but offers with the supported cipher suite list a way itself to fingerprint the device.

On the IP and TCP layers, there is also fingerprinting possible due to implementation differences. As these layers are below the TLS layer, they are completely untouched by the added encryption. Even lower on the link layer, there are also ways to do fingerprinting but these are dependent on the used link layer technology on this link. Additionally, the observer has to be on the same link as the client to do observations on the link layer.

Overall the Protocol stack of HTTPS has a lot of possibilities which can be taken into account when specific information about the clients should be examined. As fingerprinting methods are possible at very different aspects of a connection a specific attacker model is needed to say if a setup is sufficiently protected against fingerprinting.

References

[1] O. Arkin and F. Yarochkin. Xprobe v2.0 - a "fuzzy" approach to remote active operating system fingerprinting. 08 2002.

[2] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 605–616, New York, NY, USA, 2012. ACM.

[3] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. http://www.rfc-editor.org/rfc/rfc5246.txt.

[4] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, May 2012.

[5] D. Eastlake. Iana considerations and ietf protocol usage for ieee 802 parameters. RFC 5342, RFC Editor, September 2008. http://www.rfc-editor.org/rfc/rfc5342.txt.

[6] D. Eastlake. Transport layer security (tls) extensions: Extension definitions. RFC 6066, RFC Editor, January 2011. http://www.rfc-editor.org/rfc/rfc6066.txt.

[7] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. http://www.rfc-editor.org/rfc/rfc2616.txt.

[8] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[9] R. M. Gerdes, T. E. Daniels, M. Mina, and S. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *NDSS*, 2006.

[10] M. Husák, M. Cermák, T. Jirsík, and P. Celeda. Network-based https client identification using ssl/tls fingerprinting. In *2015 10th International Conference on Availability, Reliability and Security*, pages 389–396, Aug 2015.

[11] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, April 2005.

[12] R. Lippmann, D. Fried, K. Piwowarski, and W. W. Streilein. Passive operating system identification from tcp / ip packet headers *. 2003.

[13] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele. Analyzing https encrypted traffic to identify user operating system, browser and application. 03 2016.

[14] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 103–114, New York, NY, USA, 2011. ACM.

[15] V. Paxson. Automated packet trace analysis of tcp implementations. *ACM SIGCOMM*, 27, 07 2000.

[16] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc791.txt.

[17] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[18] Fyodor. Remote os detection via tcp/ip stack fingerprinting. https://nmap.org/nmap-fingerprinting-article.txt. Accessed: 2018-11-26.

[19] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 7–12, New York, NY, USA, 2013. ACM.

[20] G. Taleck. Synscan : Towards complete tcp / ip fingerprinting. 2004.

[21] F. Veysset, O. Courtay, and O. Heen. New tool and technique for remote operating system fingerprinting - full paper -. 2002.

[22] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 35–49, May 2008.

[23] M. Zalewski. p0f v3: passive fingerprinter. http://lcamtuf.coredump.cx/p0f3/README, 2012. Accessed: 2018-11-20.

Caching with Relation

Mohamad Nour Moazzen, Stefan Liebald *
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mohamadnour.moazzen@tum.de, liebald@net.in.tum.de

***Abstract*—**

The increase in network traffic across the Internet in recent days impose many challenges to web servers and Internet service providers regarding access latency and network bandwidth consumption. Web caching is one of the optimization methods used to face these challenges. The most important part of a web caching system is the cache replacement algorithm which decides which web objects should be evicted from the cache in order to make space for new ones when the cache is full. Traditional algorithms consider metrics such as recency and frequency to make the replacement decisions. In this paper we explore a type of algorithms which considers the semantics of the web objects to make the replacement decisions. We differentiate two categories of this type of algorithms according to how they interpret the contents of the web objects: subject-based and link-based. We present some algorithms of both categories and explain how they work.

***Index Terms*—WWW, Web caching, Cache replacement algorithms, Semantic distance**

1. Introduction

The Internet is a valuable source of information and it is used by an increasing number of people for education, entertainment, business and almost every aspect of modern life. This led to an increase of network traffic across the Internet, which in turn causes increasing access latency. To face these challenges, several optimization methods have been developed [1]. Web caching is one of these methods. It aims to improve the performance of web servers and decrease access latency.

We can differentiate three types of Web caching depending on the place where the web cache is employed: Client-side caching, Server-side caching and Proxy caching. In this paper we focus on proxy caching.

The basic idea of web caching is to store copies of web objects (web pages, photos, videos, etc.) which are requested by users in intermediate web caches, so future requests for these web objects will be served by these web caches instead of the original servers. This decreases the number of requests to the original servers and reduces the amount of transmitted data over the network in addition to decrease access latency because these web caches are closer to the users.

Web caching system works as follows:

- When user requests an object from a web server, the system first checks if this object is already cached.
- If there is a copy of this object in the cache it will be send directly to the user.
- If the object is not cached then it will be requested from the original web server and forwarded to the user, then this object is stored in the cache.

Because web caches have limited size, we cannot store every web object for indefinite time. When the cache is full, and new objects need to be stored in it, the system has to remove one or more cached objects in order to make space for the new objects. The decision of what objects to be removed from the cache is taken by the *cache replacement algorithm.*

Cache replacement algorithms are very crucial for the performance of the web caching system because if the algorithm removes objects which maybe requested again in the near future, this degrades the performance of the system as these objects have to be requested again from the original server. There are several types of cache replacement algorithms, each one of them depends on one or several metrics to decide which object(s) to evict from the cache.

In this paper we explore a type which depends mainly on the semantics of the contents of the web objects to make the replacement decisions. Algorithms of this type measure the relationship between the cached objects and the new incoming objects which need to be stored in cache in terms of a semantic distance calculated based on the subject of the contents or the links between objects. The objects which are the furthest from the new objects in terms of the semantic distance , i. e., the objects which are the least related to the new objects, are marked for eviction.

The rest of this paper is structured as follows. Section 2 includes a background about cache replacement algorithms. Section 3 introduces semantic cache replacement algorithms and describes several algorithms from this type. In section 4 we conclude the paper.

2. Background

Many cache replacement Algorithms have been proposed in literature.

We can differentiate between them according to the metric they use to make the replacement decisions.

Two of the most important metrics are [2]:

- Recency: time of last request for an object.
- Frequency: number of requests for an object.

Podlipnig and Böszörmenyi in [2] proposed a classification for the cache replacement Algorithms as follows:

doi: 10.2313/NET-2019-06-1_13

- Recency-based Algorithms: depend on the recency metric to make the replacement decisions. The most well-known algorithm from this category is LRU (Least Recently used), which evicts the least recently accessed objects from the cache.
- Frequency-based Algorithms: depend on the frequency metric to make the replacement decisions. The most well-known algorithm from this category is LFU (Least Frequently used), which evicts the least frequently accessed objects form the cache.
- Recency/Frequency-based Algorithms: depend on both recency and frequency metrics to make the replacement decisions. SLRU (Segmented LRU) [3] is an example of algorithms from this category.
- Function-based Algorithms: use a function to calculate the value of an object then use this value as a metric to make the replacement decisions. GD(Greedy Dual)-Size [4] is an example of algorithms from this category.
- Randomized Algorithms: these algorithms randomly make the replacement decisions. HARMONIC [5] is an example of algorithms from this category.

The most important metrics to take into account when evaluating or comparing the performance of cache replacement Algorithms are hit rate and byte hit rate.

- Hit rate: "measures the percentage of requests that are served from the cache (i. e., requests for pages that are cached)" [6].
- Byte hit rate: "measures the amount of data (in bytes) served from the cache as a percentage of the total amount of bytes requested" [6].

The algorithms which we have mentioned in this Section depend on the metadata of the cached objects to decide which ones to evict from the cache (time of last access, frequency of access, etc.).

In the next section we explore a different type of cache replacement algorithms which take into account the contents of the objects [7] when making the replacement decisions.

3. Semantics based cache replacement algorithms

Semantics based cache replacement algorithms depend mainly on semantics of the contents of the cached web objects in order to decide which ones to evict from the cache [8]. They measure the relationship between the cached objects and the new incoming objects which need to be stored in cache (or the most recently accessed cached objects).

When the replacement process is triggered, the algorithm evicts the objects in cache which are less related to the new incoming objects regarding the semantics of their contents. This type of algorithms relies on the intuition that the cached objects which are less related to the new objects regarding the semantics of their contents are less likely to be requested in the near future, therefore they can be evicted from the cache.

If the algorithm cannot decide what objects to evict because all of them are closely related to the new incoming objects, then these algorithms make use of other metrics such as recency, frequency, etc., to make the replacement decision. We can differentiate two categories of this type of algorithms according to how they interpret the contents of the web objects:

- Algorithms which calculate the semantics for the objects based on the subject of their contents (LSR [7], LSR/H [8]).
- Algorithms which calculate the semantics for the objects based on the links which are contained in these objects (SACS) [6].

In the following we describe several semantics-based algorithms.

3.1. LSR

Alcides [7] proposed an algorithm called LSR (Least Semantically Related). This algorithm relies on the assumption that every user is inclined, for a period of time, to request objects whose contents belong to a specific subject, i. e., semantically related objects. Also, it assumes that during a period of time all the objects which are requested from a cache belong to the same subject, that is, "LSR supports single thread of interest" [8].

It depends on a metric called semantic distance to make the replacement decisions. In order to calculate the semantic distance, the algorithm associates semantics to each object according to its contents.

When the replacement process is triggered, the algorithm calculates the semantic distance between each cached object and the new object(s) which need(s) to be stored in the cache. The cached objects which are less related to the new objects, i. e., the objects which have the highest semantic distance to the new objects, are marked for eviction.

According to [7], one way to associate semantics to each object is through a taxonomy, objects are organized into a tree of subjects. It starts from the root and branches out into nodes where each node represents a subject, the nodes may branch out into children nodes which represents more specific subjects, objects are distributed on the nodes according to the subject of their contents.

Using this taxonomy, we can get the semantics of any object in the form of a sequence of tree nodes from root to the specific subject which match the subject of its contents. Then the algorithm can calculate the semantic distance between any two objects by measuring the shortest path between the corresponding subject nodes.

It is hard to implement such taxonomy for the whole Internet, but there are several efforts toward this like DMOZ Open Directory Project [9] and by Schmidt et al. in [10] which proposed creating web servers which can be queried using the URL of an object to respond by its semantics.

Figure 1 shows an example of a tree of subjects, it is a real-world example represents a partial view of a taxonomy defined by the DMOZ Open Directory Project [8]. Using this figure, we can get the semantics of the object which is titled "Foundations of the Internet Protocol" in the following form: *Top.Computers.Internet.Protocols.IP*. So, for example if this object is the new object which

doi: 10.2313/NET-2019-06-1_13

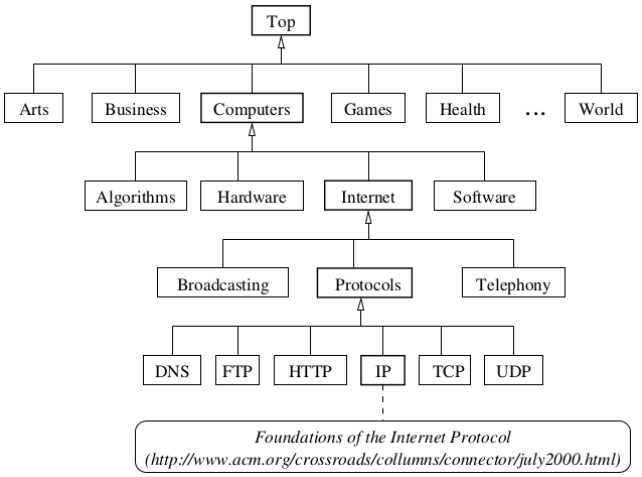

Figure 1: Partial hierarchy of subjects defined by the DMOZ Open Directory [8].

subject semantics	weight
Top.Computers.Algorithms	10
Top.Computers.Software	9
Top.Business	8
Top.Games	7
Top.Computers.Algorithms	6
Top.Computers.Internet.Protocols.IP	5
Top.Computers.Internet.Protocols.HTTP	4
Top.Games	3
Top.Business	2
Top.Computers.Internet.Protocols	1

Figure 2: A history of 10 subject accesses [8].

needs to be stored in the cache, the algorithm starts evicting objects which corresponds to subject nodes which are the furthest from *IP* node like *Art*, *Business*, *Algorithms*, etc.

3.2. LSR/H

Calsavara and Schuck [8] proposed an algorithm called LSR/H (Least Semantically Related + History of subject accesses) This algorithm is a developed version of LSR. It adds the dependency on a metric called history of subject accesses to make the replacement decision. It supports "multiple threads of interest" [8], i. e., there can be, for a period of time, multiple subjects of interest for the users. These subjects are called "hot subjects" [8]. A subject is included in the hot subjects when, for a period of time, there is a substantial number of accesses to objects whose contents are related to that subject [8].

To keep track of the hot subjects, the algorithm records the history of subject accesses. Whenever an object is accessed, it adds a weight to its corresponding subject. Recency plays a role in the weighting process, recent accesses weight more than old accesses [8]. The cached objects whose contents are less related to the hot subjects are marked for eviction, i. e., objects whose contents are related to subjects of less weight.

LSR/H, like LSR, depends on "tree of subjects" taxonomy to map semantics to objects. As we described

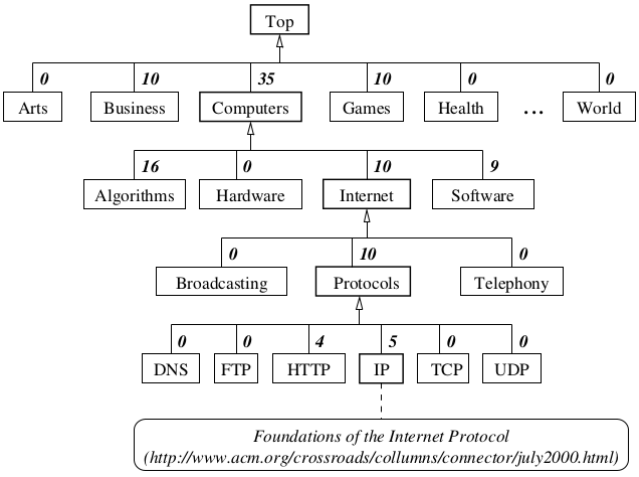

Figure 3: Hierarchy of subjects of Figure 1 with weights [8].

earlier, each request or access to an object adds a weight to its corresponding subject. In this taxonomy, when the algorithm adds a weight to a subject node, it should also add this weight to the parent nodes of this subject node recursively until reaching the root.

To clarify this, we take the tree of subjects in Figure 1 as an example. We assume that the record of history of subject accesses is of length 10, i. e., the algorithm is configured to record the subjects of the last 10 accesses. Also, we assume that when the replacement process is triggered the history of subject accesses is as shown in Figure 2.

The weight values are distributed according to the recency of access, i. e., 10 corresponds to the most recently accessed subject and 1 corresponds to the least recently accessed subject [8]. Using these values, the algorithm calculates the weights of the subjects.

For example: according to Figure 2, the weight of *Software* node is 9, the algorithm adds 9 to the node *Software* and also adds 9 to its parent node *Computers*. Figure 3 shows the tree of subjects after calculating the weights. We can see from Figure 3 that the hot subjects are mainly *Computers*, *Business* and *Games*. The algorithm starts evicting objects whose contents are related to the subjects of *Art*, *Health* and *World*, then *Business* and *Games* and so on.

3.3. SACS

André et al. [6] proposed an algorithm called SACS (Semantic Aware Caching System). This algorithm relies on the assumption that the cached objects which can be reached by links from a recently accessed object will most likely be requested in the near future, so, they should not be evicted from the cache. It depends on 3 metrics: recency, frequency and semantic distance to make the replacement decisions. The semantic distance between two objects is calculated by measuring the minimum number of links which is needed to be followed to reach one from the other [6]. SACS differentiates between two types of links when calculating the semantic distance:

doi: 10.2313/NET-2019-06-1_13

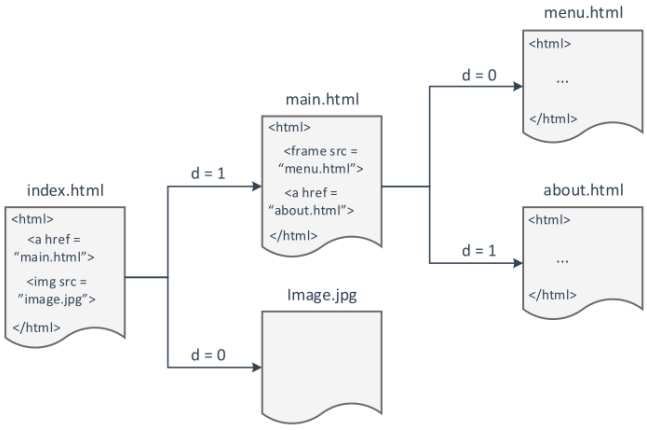

Figure 4: Example of a web site with distance assigned based on link information [6].

- Explicit link: the link that should be clicked by the user to get the referenced object. A link of this type is assigned a distance of 1 [6].
- Implicit link: the referenced object by this link is loaded automatically. A link of this type is assigned a distance of 0 [6].

Figure 4 shows an example of how SACS assigns the distance between pages of a website. SACS continuously keep track of a set of objects called pivot, they are the most recently accessed cached objects [6]. When the replacement process is triggered, the algorithm calculates the semantic distance between each cached object and the nearest member of pivot. The objects which are the most distant from pivot are marked for eviction. In case multiple objects are in the same distance from pivot, the algorithm orders them according to their frequency information (the objects which are less frequently requested are more likely to be evicted).

4. Conclusion

Web caching is one of the optimization methods used by the web servers to improve their performance and decrease access latency.

Cache replacement algorithm decides the web objects that should be evicted from the cache in order to make space for new ones when the cache is full.

Traditional algorithms consider metrics such as recency and frequency to make the replacement decisions.

In this paper we explored a type of cache replacement algorithms which considers the semantics of the contents of the web objects to make the replacement decisions. We divided it into two categories according to how they interpret the contents of the web objects: subject-based and link-based. Then we described several algorithms of this type.

According to [6], [7], [8] these algorithms perform better than the other well-known replacement algorithms such as LRU and LFU.

LSR/H builds upon LSR by adding additional metrics to improve performance.

LSR and LSR/H are hard to be implemented on the Internet scale because they need a global representation of the semantics of web objects (based on the subjects of their contents) which enables them to associate semantics to objects. This is not available for the all the objects on the Internet, though there are some efforts toward that like DMOZ open directory project [9] and by Schmidt et al. in [10] which proposed creating web servers which can be queried using the URL of an object to respond by its semantics.

SACS algorithm on the other hand does not require such condition to be implemented.

Also, LSR and LSR/H consider that each web object can be associated with only one specific subject, therefore, these algorithms are not applicable in cases where we have for example a web page or an article which discusses several subjects.

References

[1] Y. Zhang, N. Ansari, M. Wu, H. Yu, On wide area network optimization, Commun Surv Tutor IEEE 14(4):1090-1113, (2012).

[2] S. Podlipnig, L. Böszörmenyi, A survey of Web cache replacement strategies, ACM Comput. Surv. 35, 4, 374-398, (2003).

[3] R. Karedla et al, Caching Strategies to Improve Disk System Performance, Computer, vol. 27, no. 3, pp. 38-46, (1994).

[4] P. Cao, S. Irani, Cost Aware WWW Proxy Caching Algorithms, Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, CA, pp. 193-206, (1997).

[5] S. Hosseini-Khayat, Investigation of generalized caching, Ph.D. dissertation, Washington University, St. Louis, MO, (1997).

[6] N. P. André, R. Carlos, F. Paulo, V. Luis, An adaptive semantics-aware replacement algorithm for web caching. Journal of Internet Services and Applications, 6. 10. 1186/s13174-015-0018-4, (2015).

[7] C. Alcides, The least semantically related cache replacement algorithm, In Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research (LANC '03), ACM, New York, NY, USA, 21-34, (2003).

[8] A. Calsavara, M. R. Schuck, Internet object caching based on semantics and access history, Programa de Pos-Graduacao em Informatica Aplicada Pontificia Universidade Catolica do Parana Rua Imaculada Conceicao, 1155, Prado Velho 80215-901 Curitiba, PR.

[9] AOL Inc. "Archive of dmoz.org provided by Internet Marketing Ninjas", Last visited 2019-02-06, http://dmoz-odp.org/

[10] A. Calsavara, G. Schmidt, Semantic search engines, In F. Ramos, F. Unger and V. Larios, editors, Advanced Distributed Systems: Third International School and Symposium, ISSADS 2004, Guadalajara, Mexico, January 24-30, 2004, Revised Selected Papers, volume 3061 of Lecture Notes in Computer Science, pp 145-157. (2004).

doi: 10.2313/NET-2019-06-1_13

Investigating TCP SYN Flood Mitigation Techniques in the Wild

Julian Villing
Technical University of Munich, Germany
Email: julian.villing@tum.de

***Abstract*—TCP SYN flood Denial-of-Service (DoS) attacks exploit a weakness in the TCP specification. By initiating many incomplete connections, the servers' backlog is filled with the states of the half-open ones. Once there is no more space in the backlog, the server is unable to handle legitimate requests and the attack has been successful. It is important to prevent this problem as the targeted machine cannot offer its services anymore due to being unreachable. This paper introduces and compares several TCP SYN flood mitigation techniques as well as discussing challenges of their detection.**

***Index Terms*—tcp syn flood, syn cookies, syn authentication, syn flood mitigation**

1. Introduction

When establishing a new TCP connection, the server stores the corresponding state in a transmission control block (TCB) which is needed to establish the connection. The minimum size for said block is 280 bytes whereas Linux uses 1300 bytes per block. A new TCB is created and stored in the backlog for each TCP SYN packet received. This information persists during the three way handshake until the connection is either established or a timeout occures as explained by Eddy in [1].

TCP SYN flood attacks exploit this weakness: they initiate many half-open connections in a short time but never complete the handshake. The flood of connections forces the server to keep many unnecessary TCBs for those connections in the backlog which is eventually filled. Upon exhaustion, which is reached with 100 to 1000 connections, no new connections can be established to handle legitimate requests and the DoS attack has been successful [1].

Mitigating this problem is important since not being able to handle such floods can for instance lead to outages. As an example, starting in October 2012, a large US bank was a target of such a flood but unable to mitigate the attacks for more than 5 months. This caused the bank's online service to be unusable every now and then according to [10]. It is desirable that servers under attack are still capable of delivering their services. Mitigation techniques are designed for this usecase, aiming to protect the server from being unreachable.

The rest of the paper is structured as follows in Section 2, the TCP three way handshake including its weakness is explained. Different countermeasures with their advantages and disadvantes are introduced in Section 3. Section 4 covers why it is difficult to detect which of those techniques is currently in use. In the conclusion these techniques get compared in terms of memory and computing immunity as well as effectivity.

2. TCP Three Way Handshake

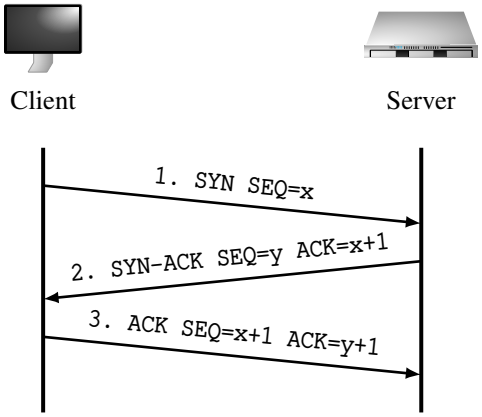

Figure 1: TCP Handshake

As seen in Figure 1, a TCP handshake consists of three packets being exchanged between the client and the server. Those are used to inform both endpoints about each other's sequence number and the corresponding acknowledgments.

When the server receives the SYN packet, a TCB is created and stored in the backlog, primarily to keep track of the options selected / requested in the header. It contains, among others, the source / destination addresses and ports as well as the sequence numbers received from and sent to the client [1].

Once the handshake is completed, the TCB is removed from the backlog and does not put stress on the bottleneck anymore. Therefore the limited resource is consumed starting with the reception of the first packet until the third packet is received or a timeout occured [1].

In case of a SYN flood, the second packet is ignored by the attacker and therefore the third packet will never be sent. As a result the TCBs use up memory until the timeouts occur [1].

3. Mitigation Techniques

This section describes and compares the advantages and disadvantages of different approaches used to decrease the vulnerability by the SYN flooding attack.

3.1. Filtering

When a router receives a packet, it verifies that the source address specified in the received packet is actually

doi: 10.2313/NET-2019-06-1_14

reachable. A reasonable check to use is strict reverse path forwarding. This method only forwards packets "if the packet is received on the [network] interface which would be used to forward [...] traffic to [the packets' source address]" as explained by Baker and Savola in [9]. The dropped packet must have a spoofed source IP and did not come from a legitimate host in the network [1] as shown by Salunkhe et al. in [3].

Advantages: Sending spoofed packets is much more difficult and no changes to TCP are necessary [1].

Disadvantages: Filtering only works for spoofed IP addresses which are not reachable and is therefore ineffective against an attacker which controls multiple hosts. Global deployment of filters is also neither guaranteed nor likely [1].

3.2. Increased Backlog

This technique solves the problem that the backlog is filled too quickly by simplyincreasing its size. The obvious result is that more connections can be stored [1] [3].

Advantages: More connections can be stored, therefore filling the backlog takes a bit longer.

Disadvantages: The backlogs' implementation is not designed to scale, e. g. the search algorithm used is not trimmed for efficiency. It can also be circumvented simply by increasing the attack rate [1].

3.3. Reduced SYN-RECEIVED Timer

Instead of increasing the backlog, this approach reduces the duration for which the connections can occupy the backlog by reducing the SYN-received timer [1] [3].

Advantages: Incomplete connection attempts get removed earlier from the backlog.

Disadvantages: If the process of establishing a connection takes longer than normal, e. g. due to a slow network connection, legitimate connections might not get established [3]. It is equally ineffective for the same reason: an increased attack rate can easily make up for the lower timeout [1].

3.4. Recycling the Oldest Half-Open TCB

This technique mitigates the attack by recycling the oldest half-open connection once the backlog is exhausted [1].

Advantages: It works if the time to fully establish a connection is lower than the time needed to fill the backlog [1]. This means that the attack rate must be low or the backlog must be big enough.

Disadvantages: The approach fails if the backlog gets filled too quickly [1] since slow connections can be evicted.

3.5. SYN Cache

Each new connection is stored in a minimized TCB which in turn is stored in a hashmap with a limited bucket size. The bucket in which to store the TCB is selected by hashing the IP adresses, ports and secret bits from the header which the server chose beforehand and the oldest entry is dropped if the bucket is full. Generating the hash from secret bits prevents malicious clients from overflowing a specific bucket and therefore dropping legitimate connections [1] [3]. It was "the most effective and the most used" technique back in 2008 according to Oncioiu and Simion in [8].

Advantages: The secret bits prevent attackers from overflowing buckets and dropping legitimate connections [1].

Disadvantages: Because the complete TCB is not stored, some information is left out and must be retransferred once the connection gets established [1].

3.6. SYN Cookies

As the value of the initial sequence number (ISN) used in the handshake can be chosen at random, the server can give this number a special meaning e. g. by encoding data.

SYN Cookies use this very number to encode the state which would otherwise be stored as a TCB in the backlog and therefore prevent the latter from filling up. The value consists of three parts which get concatenated: the slowly increasing timestamp the server keeps track of, the maximum segment size as well as a hash of the client's ISN as well as the source / destination address and port. When the client's acknowledgement is received, the server substracts one from the acknowledgement number and compares it to the encoded state using the last few timestamps. If the encoded states match, it is a legitimate client and the TCP handshake is completed successfully. The use of a timestamp prevents replaying the packet at a later time. A TCB is created using the state and directly stored as an established connection, therefore never allocating resources in the backlog [1] [3] as presented by Lemon in [5], Ricciulli et al. in [6] and Liu and Sheng in [7].

Advantages: No memory is consumed to store the state because it is encoded in the server's initial sequence number instead [1].

Disadvantages: The sequence number is smaller than the TCB, therefore, not the whole state can be stored and data retransmission may be necessary. The SYN-ACK cannot be resent because the state is not stored on the server. This behavior breaks the TCP semantics. This technique is only effective in a low degree SYN flood attack as space is traded for processing time [1].

3.7. TCP SYN Authentication

Legitimate clients can be identified if they follow the TCP specification unlike attackers who just flood the server with SYN packets.

SYN Authentication uses a special mitigation device between the client and the server. If a client wants to establish a connection to the server, it actually does the TCP handshake with that device. When the intermediate device receives a SYN packet, it responds with a SYN+ACK packet containing an invalid acknowledgment number. The client has to respond with a RST packet as defined in the TCP specification. If it does, the client is authenticated beause attackers do not handle

doi: 10.2313/NET-2019-06-1_14

Packet	Counter	Action	Pass?
SYN	None	Move to C-1	×
SYN	C-1	Move to C-3	✓
SYN	C-2	None	✓
SYN	C-3	Add to C-3	○
ACK	None	None	×
ACK	C-1	None	×
ACK	C-2	None	✓
ACK	C-3	Move to C-2	✓

✓ (pass the packet), × (drop the packet), ○ (pass is less likely)

TABLE 1: Handling of SYN and ACK packets

invalid packets and may not even receive it, e. g. if they use spoofed IP addresses. Since the client is now authenticated, the mitigation device allows direct communication between the client and the server as shown by Nagai et al. in [2]. In other flavors the server may send a reset and also track the hop count.

Advantages: The server does not allocate resources for incomplete connections as the mitigation device ensures that the connection is legitimate [2].

Disadvantages: The client needs to do the handshake twice [2]. This slightly increases the connection establishment time in times where RTTs should be low.

3.8. Three Counters

Attackers flood the server with many SYN packets without responding which can be distinguished from legitimate requests. Because the latter follow the TCP specification, they can resend packets which were lost and correctly reply to the server's responses.

This technique makes use of three counters, C-1 to C-3, in which different packets will be stored. The first counter records initial SYN packets, the second stores SYN packets of established connections, and the last records any other SYN packets. These counters are typically used after a flood has been detected. Their usage can be seen in Table 1 and is explained in the following paragraphs [4].

A 4-tuple consisting of the source / destination addresses and ports is extracted from each received SYN packet and queried against the three C-s. If it is not found, the tuple resembles a new connection and is added to C-1 while the packet is being dropped. If it is in C-1 or C-2 the packet is passed and the tuple is moved to C-3 in the first case. Otherwise the packet must be in C-3 and it is forwarded with a propability p, decreasing for an increasing number of packets received. This is achieved by adding the tuple to C-3 multiple times and querying the counter for the total amount [4].

Received ACK packets are handled in a similar manner using the same 4-tuple. If the packet is in C-2 or C-3, the packet is passed and moved to C-2 if not done yet because the connection is now completed. In any other case, the packet will be dropped because a SYN packet must be received before an ACK as explained by Gavaskar et al. in [4].

Advantages: This technique is effective against many identical packets as they are less likely to be passed the more often they are received.

Disadvantages: Every SYN packet has to be sent twice. Duplicating every SYN packet is a problem because the latter will be forwarded to the server for sure. Furthermore following the two SYNs with an ACK, without the need to actually listen for a SYN-ACK response, renders this countermeasure ineffective while not affecting other approaches because the sequence numbers are not tracked. Lastly, flooding the server with a spoofed SYN packet, essentially preventing that very packet from being forwarded can cause problems if a legitimate client attempts to connect using the same SYN packet.

3.9. Random drop

Similar to technique 3.4, a connection is dropped once the backlog is full. The difference is that the connection chosen at random and the client is informed with a TCP Reset (RST) [6].

Advantages: As most of the entries in the backlog are from the attacker, this mitigation technique has a high chance to drop the malicious connections [6].

Disadvantages: There is a small propability that legitimate connection attempts get denied [6].

3.10. SYN Agent

Instead of doing the handshake with the server, the client does that with the SYN agent instead. After the handshake is completed, there are two ways to continue depending on the kind of agent.

The first option is that the agent does the handshake with the server imitating the client. Once this is completed, the difference between the agent's and the server's sequence number has to be remembered. It is applied to each message the agent receives from either side and modified before it is forwarded.

The other option is that the agent informs the server about the successful handshake by sending an ACK packet to the server with the reserved bit set to one. This includes the sequence number the agent used while establishing the initial connection. The advantage of this method is that the agent just forwards the messages without touching them. As a result, the agent does not need to store the difference and the computational effort is also decreased [7].

Advantages: The server is guaranteed to only get to know serious connection attempts. The latter option also reduces the load on the agent [7].

Disadvantages: An extra agent is needed which has to store information about half-open connections [7]. If the first option is chosen, the agent is also given additional computational effort.

4. Detection

The detection and identification of the mitigation techniques in use is difficult because they are only active

doi: 10.2313/NET-2019-06-1_14

while the server is being flooded with excessive SYN packets. Some countermeasures are hard to identifiy from the outside even if they are currently active. To make if even more difficult, not every technique has an unique measurable outcome.

Sending SYN packets alone does not yield any information about the protection mechanisms used because the necessary information is contained within the received SYN-ACK packet. A SYN-ACK which is not sent yields additional information as well. For these information to be collected a legitimate client or tool is required which actually listens for those responses.

An **increased backlog** or the **reduced timeout** are practically undetectable because they are essentially configuration options which could also just be configured large or short respectively. Additionally, these measures simply delay the point of exhaustion by allowing more half-open connections to be stored at the same time or removing them earlier.

Recycling is mostly undetectable as the oldest half-open connection is resetted. The attack rate must be high enough for this technique to get noticable by preventing legitimate connections from being established.

Besides being quite similar, **random drop** is even harder to detect because the amount of dropped connections is equal however the legitimate clients have a decreased chance to be chosen. This technique is therefore even more difficult to be identified.

Filtering is partly detectable because SYN packets with spoofed addresses do not get forwarded and because of that no SYK-ACKs for those can be captured.

Caching is difficult to detect as it can be identified if information needs to be retransmitted. This technique should not be identifiable by dropped connections as the probability of a real one being reset is rather low.

SYN Cookies are not detectable since the initial sequence number is a hash value.

SYN Authentication is identifiable by the first SYN-ACK which is always invalid.

Three counters are detectable as the first SYN will always be dropped for a new connection. In addition sending many packets in the name of a valid client might block it from connecting.

The **agent** cannot be identified as it duplicates and imitates the server.

5. Conclusion

In order to prevent exploitation of the weakness in the TCP specification, many TCP SYN flood mitigation techniques have been developed. They are summarized in Table 2.

Some techniques like increasing the backlog or reducing the timeout focus on delaying the exhaustion of the backlog while others try to prevent it from getting full, like SYN agent or SYN cookies. Additionally, recycling and random drop take no precautions to prevent the backlog from being filled, instead, they simply drop a connection from the backlog if it is full based on the respective heuristic algorithm.

SYN Cookies are the best choice if the migitation technique must be ready for immediate use because this method is included in Linux. They can therefore be used even if no countermeasures have been taken beforehand. A SYN Agent should be used when the server must be protected from flooding attacks under all circumstances however a separate machine with enough memory handling the incoming traffic is needed. SYN Authentication can be used instead of SYN Cookies if correctly following the TCP specification is more important than the response time as the intermediate device can resent the SYN-ACK packets however the handshake has to be done twice.

Technique	Guarantee	Memory Immunity	Computing Immunity	Robustness	Good Performance
Filtering	○	✓	×	✓	×
Increased Backlog	×	×	×	✓	×
Reduced Timeout	○	×	✓	✓	✓
Recycling	○	✓	✓	✓	✓
SYN Cache	✓	✓	×	✓	×
SYN Cookies	✓	✓	×	×	✓
SYN Authentication	✓	✓	✓	✓	×
SYN Agent	✓	✓	✓	✓	✓
Three Counters	○	×	×	✓	×
Random Drop	×	✓	✓	✓	✓

✓ (fulfilled), × (not fulfilled), ○ (depends on the attack)
This table further extends the one from [6].

TABLE 2: Comparison of Mitigation Techniques

References

[1] W. Eddy, "TCP SYN Flooding Attacks and Common Migitations", RFC 4987, August 2007

[2] R. Nagai, W. Kurihara, S. Higuchi, T. Hirotsiu, "Design and Implementation of an OpenFlow-based TCP SYN Flood Mitigation", 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2018

[3] H. S. Salunkhe, Prof. S. Jadhav, Prof. V. Bhosale, "Analysis and Review of TCP SYN Flood Attack on Network with Its Detection and Performance Metrics", International Journal of Engineering Research & Technology, January 2017

[4] S. Gavaskar, R. Surendiran, Dr. E. Ramaraj, "Three Counter Defense Mechanism for TCP SYN Flooding Attacks", International Journal of Computer Applications, September 2010

[5] J. Lemon, "Resisting SYN flood DoS Attacks with a SYN Cache", USENIX Association, February 2002

[6] L. Ricciulli, P. Lincoln, P. Kakkar, "TCP SYN Flooding Defense"

[7] P.-E. Liu, Z.-H. Sheng, "Defending Against TCP SYN Flooding with a new Kind of SYN-Agent", Proceeding of the 7th International Conference on Machine Learning and Cybernetings, July 2008

[8] R. Oncioiu, E. Simion, "Approach to Prevent SYN Flood DoS Attacks in Cloud"

[9] F. Baker, P. Savola, "Ingress Filterin for Multihomed networks", RFC 3704, March 2004

[10] radware Inc, "Operation Ababil", Online, 2013, last visited 2018-12-06, https://security.radware.com/WorkArea/DownloadAsset.aspx?id=848

doi: 10.2313/NET-2019-06-1_14

Networking in Biscuit

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

***Abstract*—Biscuit is a High Level Language (HLL) kernel written in Go as a research project to evaluate the impact HLLs have on the performance of operating systems. Go was chosen as it allows easy asm calls, offers good concurrency and can be statically analyzed and compiles to machine code. While Biscuit as a whole has been thoroughly analyzed, this is not true for its components. In this paper we present Biscuit's ixgbe network driver and its implementation. The driver is separated into three main parts: initialization and packet reception and transmission. Afterwards it is compared with ixy.go, an ixgbe user space network driver for Linux systems. Go systems are notably slower (around 15%) then the same systems written in C but offer increased security as memory related bugs cannot occur.**

***Index Terms*—Biscuit, Go, Networking, NIC, Driver**

1. Introduction

Today the internet is arguably one of the most important aspects of IT. The ISO/OSI model is well known and describes the process of sending information from one participant to another on a high and abstract level. Network Interface Controller (NICs) are pieces of hardware that implement the necessary functionality to send and receive data using physical and link layer standards and thus provide the base of the ISO/OSI model. NICs are accessed using drivers that allow communication between the physical card and the software running on the computer and therefore are an integral part of Operating System (OS) kernels. These kernels are almost exclusively written in C (or the C family) and assembler, be it a Linux kernel or a Windows NT kernel and thus the drivers are as well. However in recent years there has been an effort to write OS kernels in different languages. Biscuit [1] is such a kernel and is written in the Go programming language and assembler. In this paper we will take a look at its implementation of the ixgbe network driver for the `Intel 82599 10GbE` controller.

Section 2 gives an overview of network drivers in general, more specifically of those written in Go, and the resulting differences. In Section 3 the driver implemented in the Biscuit kernel will be presented. Next we will compare the driver with a user space network driver for the same device family written in Go, ixy.go, in Section 4 and finally, close this paper by pointing out advantages and disadvantages of these drivers in section 5.

2. Network Drivers (in Go)

Drivers are pieces of software that allow communication between a hardware device and the rest of the system. The driver abstracts from the actual hardware access and instead exposes interfaces that are better to handle from an outside programmer's perspective. The datasheet for the `Intel 82599` family is freely available online and describes the NIC in its full extent [2].

In case of drivers for network cards there are multiple things that have to be managed:

- Transmit queues (TX) and receive queues (RX) are used by the NIC to receive and send packets. They are organized as ring buffers.
- DMA (Direct Memory Access) memory for the packets which the NIC and the driver can access.
- DMA memory for the packet descriptors. They hold information about the corresponding packet buffers and the packets contained in them and control the behavior of the packet buffers. They are organized as ring buffers.
- Access to the PCI device file in order to control the NIC and enable DMA memory.
- Access to the pagemap (interface for the page tables) as NICs work with physical addresses. It has to be ensured that the mapping stays consistent.

These are general problems for network drivers, independently of whether they are written for the user or kernel space. Specific problems are discussed in Section 4.

Using Go to write network drivers is similar to writing a network driver in C, as the language has intentionally been designed with this similarity in mind. Apart from obvious changes to the syntax it has to be noted that Go does not support many of the tools that are standard in systems programming, specifically in driver programming. These include the volatile operator as reads and writes to registers have to be processed immediately and cannot be cached. Another important aspect of Go is that while it does support pointers, due to its runtime and type safety feature, it does not support pointer arithmetic. In C one usually operates on the allocated DMA memory via pointers. In Go this is only possible via the use of the unsafe package that offers unsafe or arbitrary pointers, which circumvent the runtime as well as many safety measures. Thus care is required in order to break as few assumptions of the runtime as possible when using this approach.

doi: 10.2313/NET-2019-06-1_15

3. Architecture

In the following we will take a look at the architecture of biscuit's network driver. We start with a high level description and then go on to further elaborate the details and components. All function names, line numbers, and other code references refer to the file `/biscuit/src/ixgbe/ixgbe.go` of commit `2a2dbe1` of the biscuit github page [1].

3.1. Overview

On a high level the driver manages the NIC and its state, transmit and receive queues, handles incoming packets and hands them over to applications on top of the driver, and offers an interface which can be used to send packets.

The driver can be loaded by calling its `Ixgbe_init()` function, which registers the PCI device and its initialization function `attach_ixgbe()` (lines 1272ff.). This function executes the setup procedure: configure flow control, offloading, RX and TX queues and interrupts. After the initialization, received packets will automatically be handed over to the network stack and an interface `Tx_[raw|ipv4|tcp|tcp_tso]()` (lines 911ff.) is provided for sending packets of different types.

3.2. Details

After this high level view of the driver's architecture we present a more detailed description of its components. It is recommended to have the implementation and the datasheet at hand. References to sections in the datasheet will be noted as `DS X` where X is the section number.

3.2.1. Receive and Transmit Descriptors. The code itself starts with the definition of all relevant constants and corresponding helper functions (lines 1 to 309, DS 8.2 and 8.3). Afterwards the TX and RX descriptors and functions on them are defined. Note that the advanced descriptors are used. Refer to DS 7.1.6 for the Advanced Receive Descriptors and DS 7.2.3.2 for the Advanced Transmit Descriptors. These registers are represented as two 64-bit unsigned integers each, thus reading and writing is done by setting the corresponding bits to 0 or 1. The NIC supports various offloading features such as computing and verifying checksums. The function `ipsumok()` (line 356) operates on the write-back format and checks, in case a non layer two packet was received, whether the bits 6 and 31 of the descriptor's second line are set as this indicates a bad IP checksum.

3.2.2. Device Properties. Next, starting from line 615, the status of the device is described and functions are defined that handle and mutate its state. A device has the following properties:

- Pci address (tag):
  The pci address the device is located at. Most importantly the Base Address Registers (BARs) addresses are exposed via pci.
- BAR0 address (bar0):
  The BARs expose configuration and control registers to the drivers. While the NIC's address space is mapped into multiple memory regions, only the BAR0 is necessary as described in DS 8.1. This address space is mapped in `init()` (line 650ff.) and accessible via the bar0 slice.
  The functions `rs()` and `rl()` (lines 681ff. and 688ff.) write to and read from the registers as an offset of bar0.
- Transmit queues (txs):
  Queues that are used for packet transmissions. Contains TX descriptors and their current number. The queue tail as well as some additional parameters are cached to reduce expensive register reads.
  Packets can be enqueued for transmission with the `Tx_[raw|ipv4|tcp|tcp_tso]()` (lines 911ff.) functions. Note that sending is asynchronous: an enqueued packet does not have to be sent out immediately but the NIC will set the DD flag of the TX descriptor once it has been sent out.
- Receive queue (rx):
  Queue that are used for packet receptions. Contains RX descriptors, their current number, a slice referencing the packets and the queue tail is cached.
- Number of allocated pages (pgs):
  Incremented by one whenever a page is added (see `pg_new()`, line 89ff.).
- Link status (linkup):
  Whether the NIC is operational.

This is not the full list but includes most that are relevant for a general understanding. Refer to the implementation for the full list.

3.2.3. Sending. Next we will take a more detailed look at sending. The `Tx_[raw|ipv4|tcp|tcp_tso]()` functions (note that these are the exported functions as they start with a capital letter) all call `_tx_nowait()` (lines 927ff.) with the corresponding arguments. This function locks a TX queue and calls `_tx_enqueue()` (lines 963ff.) which handles the actual sending. It takes information about the packet to send and tries to enqueue the packet in the transmission queue. The function returns true on success.

First the packet buffer is checked for empty rows which will be deleted and parameters are checked for correctness. The hardware controls the head pointer and the driver the tail pointer of the ring buffer. The next step is to find out how many buffers are needed and whether there are enough that are free for use. The DD (descriptor done) flag of the status register is set when the descriptor is done, indicating that the packet has been sent out and the descriptor can be reused. The eop (end of packet) flag is set if it is the last descriptor of a packet (see DS 7.2.3.2.4 for the flags). If there are not enough descriptors for the packet buffer, the function returns false, else there is enough space in the transmit queue and the packet can be enqueued.

For sending, the packet headers will be handled first and the rest of the packet afterwards, refer to lines 1046 ff. for the enqueueing. Depending on the type of packet (ethernet, ipv4, etc.) the approach has to be different to accommodate to the packet properties. After the header is done, the payload can be treated independently. The

1. https://github.com/mit-pdos/biscuit/tree/2a2dbe1228881c94764f1cdf6134dca27defab12

doi: 10.2313/NET-2019-06-1_15

only thing left is to update the tail pointer so that the NIC can now send out the packets in the queue.

3.2.4. Receiving. Now that we have discussed how sending packets works, the next step is receiving packets. This is implemented in `rx_consume()` in lines 1086ff. Receiving works on a RX queue which is organized similar to a TX queue. A memory area that is handled as a ring buffer with a head pointer controlled by the hardware and a tail pointer controlled by the driver. When processing received packets it first has to be checked for the DD flag, which indicates a received packet, until the current head is found. In the case of no newly received packets the function is done. Note that the tail itself is empty and thus has to be skipped. For each descriptor DMA memory in the size of the packet is allocated and the packet is then handed over to the network stack. Afterwards the descriptors are reset and the tail pointer update is sent to the NIC.

3.3. Interrupt Handling

When operating a NIC interrupts may be triggered which need to be handled by the driver. For this driver interrupt handling is implemented in int_handler() (lines 1151ff.). Interrupts are described in DS 7.3. First the interrupt handle has to be registered. The code then runs in a forever loop: wait for an interrupt and handle it.

Four different types of interrupts in the Extended Interrupt Cause Register (EICR, DS 7.3.1.1 for the description and DS 8.2.3.5.1 for the register) are handled. Note that the queue interrupts are mapped to bit 0 for all RX queues and to bit 1 for all TX queues (lines 1406-1409, DS 8.2.3.5.16).

- RX queue interrupt (bit 0):
  Is raised on descriptor write back, a full queue or upon reaching a minimum threshold. Thus as this indicates newly received packets, they are to be retrieved via a call to `rx_consume()`
- TX queue interrupt (bit 1):
  Is raised on descriptor write back. This indicates that packets have been sent out but as sending is done via the corresponding functions, nothing has to be done. It has to be assumed that this is left over from the programming process.
- Rx Miss (bit 17):
  Is raised when packets are dropped due to a full Rx queue (overrun). Nothing additional that can be done as rx_consume() would already be running and packets arrive faster than they are being sent out; increment statistic.
- Link Status Change (bit 20):
  Is raised when the link status changes, e.g. from down to up or vice versa. The new status is printed, the NIC is tested and a goroutine is started that periodically prints the number of received and dropped packets.

3.3.1. Setting Up the NIC. Now that the operations of the driver on the NIC are defined, the only thing that is left is the setup of the device. This is done in the `attach_ixgbe()` function (lines 1272ff.). DS 4.6.3 describes this procedure. Please refer to the datasheet for flag and register names and other details which we cannot cover here. Note that the steps in the driver are not necessarily is the same order as proposed in the datasheet as reordering can be more efficient, as long as it does not influence the result e.g. PHY is reset before waiting for the DMA initialization as the latter has no influence on the former.

1) Disable Interrupts and call `init()` (lines 650ff.) on a new `ixgbe_t` struct which from then on represents the device. After the reset disable interrupts again (DS 4.6.3.1).
2) As flow control is disabled, the registers FCTTV, FCRTL, FCRTH, FCRTV and FCCFG are set to 0x0 (DS 4.6.3.2) and the assumption of disabled flow control is checked.
3) No snoop is enabled. Processor caches do not have to be snooped in this case and direct access to the DRAM is faster.
4) The physical address is reset via MDI command: the MSCA register (8.2.3.22.11) allows the use of the MDIO interface (3.7.6) with which physical registers can be accessed. As clause 45 operations are utilized, op code `00b` has to be sent first and afterwards `11b` for the read (DS 3.7.6.4).
5) Wait for the DMAIDONE flag of the RDRXCTL register.
6) Load the MAC address from the Receive Address Registers and cache it.
7) Enable Message Signaled Interrupts (MSI) via PCI and ensure that legacy interrupts are disabled. Reset all interrupts (EIAC register), disable automask (EIAM registers), disable interrupt throttling (EITR(n)), and map all RX queues to EICR bit 0 and all TX queues to EICR bit 1 (4.6.6).
8) Disable Receive Side Coalescing (RSC), a technique that would accumulate TCP/IP packets that belong to the same flow into large packets [3].
9) Enable and configure receive queues (4.6.7):
   a) Disable VLAN features PFVFSPOOF, MPSAR, PFUTA, PFVLVFB (4.6.10) and VFTA (7.4.4).
   b) Enable ethernet boardcast packets for ARP functionality (FCTRL bit 10).
   c) Offloading: IP checksum (RXCSUM bit 12, 8.2.3.7.5), strip CRC (RDRXCTL bit 1) and bit 12 of the DCA control register must be set to 0.
   d) Setup RX queue:
      i) Allocate new page for queue and send address and size to NIC.
      ii) Calculate number of descriptors and allocate a new packet buffer for each (2048B buffers, two per page).
      iii) Disable header splitting (SRRCTL bits 25:27), write descriptors to the NIC and initialize the receive head pointer (RDH).
      iv) Enable the queue (RDRXCTL bit 25), set the receive tail pointer (RDT) and enable receive (RXCTRL bit 0).
10) Enable and configure transmit queues (4.6.8):
   a) Map all TX queue statistics to a single counter.
   b) Enable layer two offloading via HLREG0 (7.1.3): CRC offloading (bit 0) and stripping (bit 1), padding (bit 10) and receive length errors (bit

doi: 10.2313/NET-2019-06-1_15

27).

c) `_dbc_init()` (actually referring to DCB, Data Center Bridging, see 4.6.11 for the configuration and 7.7 for the description) (lines 1779ff.):
   - Implements DCB-off, VT-off (4.6.11.3.4) as neither flow control not virtualization is supported.

d) Setup multiple TX queues (default four):
   i) Number each queue, allocate a new page for the descriptors and send address and size to NIC.
   ii) Calculate number of descriptors and allocate a new packet buffer for each (2048B buffers, two per page) and set the DD and eop flags so they are ready for use.
   iii) Disable head write-back of the queue.
   iv) Transmit Control (TXDCTL(queue_id)): Set thresholds for Pre-Fetch, Host and Write-Back. These values orient themselves at the number of descriptors that fin in a cache line to avoid cache thrashing.
   v) Initialize descriptor head and tail.

e) Enable transmission (DMATXCTL bit 0).

f) Enable transmission queues (TXDCTL(queue_id) bit 25) and wait for success.

11) Configure and enable interrupts:

a) Set the General Purpose Interrupt Enable register (GPIE, 8.2.3.5.18) to 0. This, among others, configures the use of MSI interrupts and clears the EICR register on read and disables many unneeded features.

b) Set the interrupt throttle to a 125μs as lower values can have significant impact on performance, especially within TCP bulk transfer.

c) Clear previous interrupts (EICR) and start the interrupt handler as a goroutine.

d) Enable transmit and receive queue interrupts as well as link change interrupts while disabling all other types of interrupts (EIMS).

The rest of the code are testing functions which we will not discuss in this work.

4. Comparison with ixy.go

Another driver for `Intel 82599 10 GbE` Controller written in Go is ixy.go [4]. This is a user space driver for Linux operating systems. This means that it runs completely in user space compared to Biscuit's ixgbe driver that is part of the kernel. Therefore while the NIC is still programmed in the same way, the approach differs at times. We will not consider differences in the functionality of the drivers as ixy.go is meant to be an educational driver and thus is intentionally kept simple and without much of the functionality that a driver for a running kernel needs. From a high point of view, there is not much difference to be found: On startup the NIC is initialized by programming the registers. Afterwards received packets are handled and packets can be send via an interface. Table 1 lists high level stats of both systems. However writing a kernel driver versus writing a user space driver imposes two mayor differences:

1) A user space driver does not have access to privileged system functions and must use syscalls instead.
2) While Biscuit's driver has to provide general purpose packet processing by itself for the rest of the system, ixy.go offers an API with explicit memory allocation, batching and abstraction that is similar to DPDK [5]

Two main challenges arise from the first point. As new pages cannot simply be allocated from the user space, this has to be done via Mmap() from the syscall package [6]. This also changes the way memory is administered. The second challenge is the page virtualization. In Biscuit's driver a new physical page is allocated but from the user space only virtual pages can be allocated. The mapping virtual to physical addresses for the NIC via the pagemap is not an issue but the page migration algorithm can change this mapping at any time. Fortunately, it is not implemented for huge pages which are thus used to keep the physical addresses consistent.

The second difference is mainly an architectural one. Biscuit's network driver automatically checks for incoming packets upon interrupts and hands them over to the network stack. ixy.go on the other hand offers the RxBatch() function which checks for received packets and hands them back in the provided buffer. It is the responsibility of applications built on top of ixy.go to regularly check for incoming packets instead of the driver. Sending is more similar with the main difference being batching. ixy.go has a virtual copy of the packet ring as reading flags is a rather expensive operation. When sending a batch of packets, it is first checked whether a batch of packets has been sent out, reducing the register access to once per batch and afterwards enqueueing as many packets as possible. Biscuit's driver instead checks the descriptor once per previously sent packet (eop is cached) until enough space is found but does so for each packet.

5. Conclusion

In this paper we took a look at the network driver of the biscuit kernel. Biscuit has been developed as a scientific operating system to test the impact of higher level languages on operating system kernels. While the system runs notably slower due to the garbage collection and runtime, this impact might still be acceptable for certain use cases. On the other hand Go also brings a distinct set of advantages such as memory safety. Advantages as well as disadvantages also extend to the network driver. Here speed is key and Go is slower then C. The user space driver ixy.go [4] showcases a clear loss in performance compared to its parent project written in C (10-20% depending on batch size and CPU speed). Cutler et al. also found that the kernel as a whole suffered a performance loss of 15% [1] compared to a C kernel. Still, speed is not everything, a driver also has to be secure. Cutler at al. analyzed 65 code execution vulnerabilities in the Linux kernel, 40 of which would have been prevented when using Go. In the end the choice of language always results in a trade-off between speed and security.

Because of these trade-offs we argue that is important to re-implement existing software in other programing languages. In this case Go offers security mechanisms where C programs are vulnerable. Re-implementations can be evaluated and their advantages and disadvantages

doi: 10.2313/NET-2019-06-1_15

	biscuit	ixy.go
lines	1900 (1600 without register offsets)	1000
unsafe pointer	rx & tx descriptor structs	register access, physical address calculation
memory allocation	physical pages	mmap(2) syscall
application area	provide packet receive & send functionality to kernel	low level API for fast packet processing

TABLE 1: Comparison of Biscuit's ixgbe driver and ixy.go

quantified. Similar to the CAP theorem [7] there are always properties that are more important then others depending on the system. Thus having the same programs with different properties lead to generally better systems.

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)*, (Carlsbad, CA), pp. 89–105, USENIX Association, 2018.

[2] Intel, "Intel 82599 10 gbe controller datasheet rev. 3.3." https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf. Last visited 30.11.2018.

[3] S. Makineni, R. Iyer, P. Sarangam, D. Newell, L. Zhao, R. Illikkal, and J. Moses, "Receive side coalescing for accelerating tcp/ip processing," in *High Performance Computing - HiPC 2006* (Y. Robert, M. Parashar, R. Badrinath, and V. K. Prasanna, eds.), (Berlin, Heidelberg), pp. 289–300, Springer Berlin Heidelberg, 2006.

[4] S. Voit and P. Emmerich, "Writing network drivers in go," 2018.

[5] DPDK Project, "Dpdk website." https://dpdk.org/. Last visited 14.12.2018.

[6] Go Project, "Go syscall package." https://golang.org/pkg/syscall/. Last visited 14.12.2018.

[7] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent available partition-tolerant web services," in *In ACM SIGACT News*, p. 2002, 2002.

Recent Progress on the QUIC Protocol

Mehdi Yosofie, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mehdi.yosofie@tum.de, jaeger@net.in.tum.de

***Abstract*—Internet services increase rapidly and much data is sent back and forth inside it. The most widely used network infrastructure is the HTTPS stack which has several disadvantages. To reduce handshake latency in network traffic, Google's researchers built a new multi-layer transfer protocol called Quick UDP Internet Connections (QUIC). It is implemented and tested on Google's servers and clients and proves its suitability in everyday Internet traffic. QUIC's new paradigm integrates the security and transport layer of the widely used HTTPS stack into one and violates the OSI model. QUIC takes advantages of existing protocols and integrates them in a new transport protocol providing less latency, more data flow on wire, and better deployability. QUIC removes head-of-line blocking and provides a pluggable Congestion Control interface.**

This paper indicates the disadvantages of the traditional HTTPS stack and presents the main features of the QUIC protocol which is currently standardized by the Internet Engineering Task Force (*IETF*).

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

1. Introduction

The Internet is used every day by many readers of this paper. Internet giants like Amazon, Google, and Facebook provide many applications and (web) services which let the amount of data grow significantly. This data has to be exchanged fast, reliably, and securely. Transferring this data is possible through the existing infrastructure. The most widely used one is the HTTPS stack. HTTP is transported over TCP and is secured through TLS. This network paradigm is approved and will be used for many years to come. However, the speed to built up a connection between client and server and to deliver data between them can be improved. Quick UDP Internet Connections (QUIC) may be a solution and a good replacement of the traditional Internet stack. The new paradigm of QUIC combines the transport layer and the security layer into one and provides improved features than TCP/TLS. QUIC is developed by Google. The researchers implemented the protocol and tested it on production mode on their servers such as YouTube and other Google web services, and client applications such as Chrome/Chromium. To be able to communicate over QUIC, both server and client have to provide a QUIC implementation. Thus, QUIC requires client support on application level like in the browser. QUIC aroused the interest of the Internet Engineering Task Force (*IETF*) and is on standardization progress. The *IETF* is an Internet committee which deals with Internet technologies and publishes Internet standards. Currently, QUIC is being standardized, and it remains to be seen, how it will influence the Internet traffic afterwards.

The rest of this paper is structured as follows: Section 2 presents background information about the established TCP/TLS stack needed for the problem analysis. Section 3 explicitly investigates some QUIC features like stream-multiplexing, security, loss recovery, congestion control, flow control, QUIC's handshake, its data format, and the Multipath extension. They each rely on current *IETF* standardization work, and are compared to the traditional TCP/TLS stack. In Section 4, related work about comparable protocols like SPDY or SCTP, and other work of *IETF* is discussed. Section 5 concludes this paper.

2. Background

The Transmission Control Protocol (TCP) is the standard transport protocol that most applications are based on. Data transport over TCP is reliable and packets are organized in an ordered way. Further, lost data is identified and delivered again. Other TCP features are congestion control and flow control which are necessary to not overload the receiver or the network. All of these concepts are relevant and necessary in the transport layer so that application level protocols like HTTP do not have to care about and it is ensured that application level data are not missing on endpoints. In the traditional web stack, TCP is used as the underlying protocol to transport HTTP data. To refer to the OSI model, transferring HTTP over TCP is additionally protected by TLS. TLS is a security setup which is on top of TCP (compare Figure 1). Although it is the widely used transport protocol to transmit data reliably, TCP has several disadvantages.

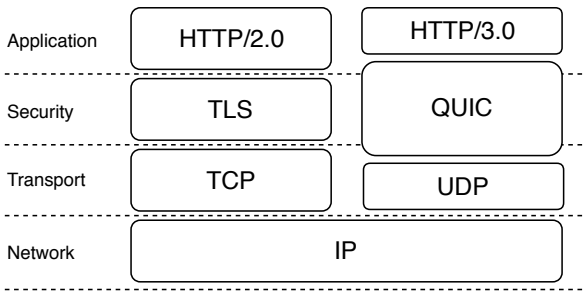

Figure 1: Traditional network stack and QUIC in comparison, adapted from [1]

doi: 10.2313/NET-2019-06-1_16

It is more difficult to have a faster development cycle and publish new releases of kernel based implementations like TCP. This protocol is implemented as part of the kernel in an operating system, i.e. network connections based on TCP run in kernel mode. Devices often have to be upgraded on both client and server side. Bringing changes to TCP and thus to the kernel would in turn cause changes to the operating system. Moreover, TCP has some inconveniences like the head-of-line-blocking delay and the handshake delay. To transfer data, at least one round trip is needed to set up the TCP connection. Furthermore, the security layer with TLS adds two further round trip delays on top TCP connections (compare Figure 1). Even if the handshake delay seems to be solved with TLS 1.3 and TCP Fast Open, the data transfer can still be optimized.

To reduce handshake latency, there is a new approach. QUIC runs – in contrast to TCP – in user space, and uses the User Datagram Protocol (UDP) as the underlying transport protocol. UDP is a widely used and lightweight transport protocol. Thus, it is suitable to be used as transport layer protocol to transmit data from host to host. The advantage of UDP is that it can traverse middleboxes like firewalls. Many firewalls, especially in big companies, block unknown protocols. Thus, unfamiliar packets could be dropped by middleboxes. QUIC encrypts and authenticates its packets and makes it possible to be transported by UDP. There is a higher probability to get UDP packets through the Internet. However, because of the required, but missing TCP features in UDP, QUIC has to implement everything that makes the protocol safe, secure, fast, and reliable by its own: QUIC has to implement congestion control and flow control. It has to define its handshake, and, concerning security reasons, it also should decide for an algorithm about encryption and authentication. These features are implemented by QUIC on the application layer. Additionally, a reason why TCP is not used, is its slow development cycle. This makes it easier to decide for the data transport protocol UDP. Thus, it is more comfortable to bring new releases and adapt features of QUIC without concerning the long term development cycle of the kernel based TCP.

TCP's handshake is the well known *Three-Way-Handshake* (RFC 793). It takes one round trip until data can be sent. If the current overall used TLS version, TLS 1.2 is used on top of TCP for encryption and authentication, two more round trip delays come additionally according to RFC 5246. This would sum up to three round trips until payload can be sent. For the QUIC standardization, the new version 1.3 of TLS will be used. Because QUIC puts encryption and transport together into the same layer in user space, it is possible to overlay the key exchanges of encryption and transport, and have a 0-RTT handshake to same servers. TLS 1.3 is already an *IETF* standard described in RFC 8446. However, hitherto this protocol is barely implemented, however, it will be the standard security protocol from the near future on. Ubuntu announced in its new interim release 18.10 the updated OpenSSL package which is equipped with TLS 1.3. Some applications use TLS 1.3 as default on the new OS version. According to [2], in the next Ubuntu release, TLS 1.3 will be used by more applications.

Users could firstly use QUIC through Google applications like the Chrome browser, the open source version Chromium, and the YouTube application on Android [1]. Gradually, there are more and more both client and server side supports. The Opera browser also supports QUIC if the corresponding QUIC flag is enabled. Except that YouTube and all other web based Google services have server side QUIC support, there are other implementations in Go and C/C++ [3], [4] which are partly used in the Caddy web server, and the LiteSpeed web server respectively. An implementation in Rust is also available [5]. Like every other network protocol, QUIC has to be implemented on both server and client to interact with each other. Other browsers and (web) servers could begin to support QUIC after the standardization is finished by the *IETF*. Since Google controls both client and server side of applications, it can implement and deploy such protocols and test it on their servers and world wide used client applications. Google's leading position in the network technology supports them to develop such protocols.

If the QUIC flag is enabled in Chrome and a client does a request via TCP and TLS, the QUIC server advertises with a QUIC flag in his HTTP answer. The next client side request, if the client wishes, will be a race between TCP/TLS and QUIC [1]. The faster reply will be the protocol stack which will be used for that request. QUIC will only be chosen if in the whole path from client to server QUIC is enabled and supported. Chrome and Chromium users can currently set the corresponding QUIC flag in their browsers to activate QUIC. Servers and companies with production based services may disable UDP inputs by their firewalls due to UDP spoofing attacks. In those intranets, QUIC can currently not be used.

3. Standardization

This Section describes the most essential QUIC features such as security, loss detection, congestion control, flow control, QUIC's data format, its handshake, and the Multipath extension. While QUIC was developed as a monolithic infrastructure by Google [1], the *IETF* work is modularizing it into separate parts. Some details such as the data format will be changed, although the core and the paradigm of QUIC will be the same as initiated by Google. Most of the concepts explained in the next Subsections, are planned to be standalone RFCs [6].

3.1. Stream Multiplexing

To fetch data fast and in parallel, HTTP/1.1 opens multiple TCP connections (compare Figure 2a). However, since each single connection has to be handled, this approach may be inefficient and may cost high CPU rates on constrained devices. HTTP/2.0 proposed to use a single TCP connection, but multiple streams. On every stream, data can be delivered (Figure 2b). The problem of this approach, due to TCP's in order delivery, is the head-of-line-blocking delay. If a packet of one stream is lost, then all other streams are blocked. The head of the line is blocking the whole connection. QUIC allows multiple streams like HTTP/2.0 but does not block all other streams due to a blocking stream (Figure 2c). Data delivery on other streams are not postponed because UDP is not bound on the in-order delivery. According to the latest core protocol draft, streams are identified by a unique stream

doi: 10.2313/NET-2019-06-1_16

ID and data delivery ordering is handled with stream offsets within a stream.

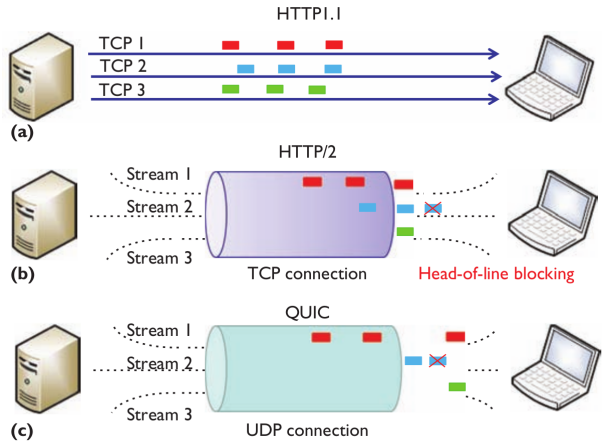

Figure 2: The different HTTP versions and the upcoming HTTP/3.0 in comparison [7]

3.2. Security

Encrypted packets play a key role in Internet traffic. Thereby, network attacks like packet sniffing or man-in-the-middle-attacks can be reduced or even stopped. Only authorized entities should be able to read specific packets in their original formats. This means, QUIC as a secure transport protocol has to provide confidentiality, integrity and authentication.

The first launch of the QUIC development was around 2012 by Google. The researchers implemented their own cryptography for QUIC to reach the goal to exchange payload with zero round trip delay. Since TLS 1.3 is deployed as standard, the *IETF* working group decided for TLS 1.3 as the security layer for QUIC. TLS 1.3 provides a 0-RTT handshake to known servers and thus fits to QUIC's target to reduce handshake lateny.

According to the current Chromium testable implementation, a QUIC server needs to have a valid certificate, and a private key in correctly supported formats to run [8]. TLS is integrated into QUIC handshakes. The current document draft about QUIC-TLS mentions that a server must have a certificate signed by a valid certificate authority, and the client must authenticate the identity of the server during the handshakes. This would mean that even a test server has to install a certificate to be runnable. The positive effect would be a tendency to the overall HTTPS usage and thus, to secure and authenticate web servers in the Internet.

3.3. Loss Recovery

QUIC packets always have increasing packet numbers and same packet numbers do not occur in a number space. This concludes that packets with higher packet numbers are sent later than packets with lower packet numbers. This way, there will not be the retransmission ambiguity problem as in TCP. In TCP, if a packet is sent, and no ACK is received within a timer, the same packet is sent again with the same sequence number. If an ACK arrives, it cannot be determined, which packet is acknowledged, and the round trip time is not measured reliably. With monotonically increasing sequence numbers, the RTT can be determined more accurately and there is no ambiguity problem. Further, regarding loss recovery, a packet which is declared lost, will have a new sequence number and is sent again [9].

3.4. Congestion Control

To reduce and avoid high network load when too many packets are sent by many endpoints in short times and packets cannot pass on, there is the need of congestion control algorithms to control the network rate and reduce the network load. The Google implementation of QUIC is adjustable so that any congestion control algorithm can be experimented and may work. During the development and testing cycle, Cubic was used as congestion controller by the Google developers [1]. Cubic is also the standard congestion controller of the Linux TCP [10]. The first *IETF* draft concerning congestion control in QUIC also mentions that Cubic was the default congestion controller. At that time, Reno was another option to use, since it is fully implemented [9]. In the latest draft of the *IETF* working group about congestion control, NewReno is announced on what QUIC bases on. However, it is also emphasized that every host and every implementation may use a different congestion controller, since QUIC supports a pluggable congestion control interface.

3.5. Flow Control

Flow control is a data transport feature not to overload the receiver. If an endpoint cannot receive data as fast as the sender sends, the receiver is overstrained, cannot handle all incoming packets and drops packets. Flow control mechanism handles the data flow between the sender and receiver so that the receiver does not get more data than its buffer can store. QUIC has, similiar to HTTP/2, two levels of flow control: stream-level flow control and connection-level flow control. Stream flow control limits the data flow for every single stream so that a specific stream will not be able to claim the entire receive buffer for itself and to preclude other streams which send data on other streams. Connection flow control means, the sender does not exceed the receiver's buffer on a connection for all streams. However, it works the same way as stream flow control, for the entire connection.

3.6. Handshake

QUIC distinguishes between the 1-RTT and the 0-RTT handshake. The cryptographic handshake minimizes handshake latency by using known server credentials on repeat connections. A client stores information about the server and can use the 0-RTT handshake on subsequent connections to the same origin. The 1-RTT handshake is possible because the transport and cryptographic keys are overlapped into the same layer. The 0-RTT handshake is possible, since TLS 1.3 provides a 0-RTT and TLS 1.3 will be used as security layer in QUIC. Since many of the network connections are to same servers which

doi: 10.2313/NET-2019-06-1_16

were contacted before, the 0-RTT makes it possible for a client to send payload without repeating cryptographic or transport key exchange.

3.7. Data Format

The data format and the naming conventions of QUIC packets and its fields are repeatedly in change during the standardization. The current draft distinguishes between a long header packet and a short header packet. The long header packet has different types: *Initial*, *Retry*, *Handshake*, and *0-RTT Protected*. Thus, the initial connection between two communication partners is established using the long header. After connection establishment, the short header is used. All types of packets ensure confidentiality and integrity [11].

Furthermore, according to the latest draft, there is the *Version Negotiation Packet* which seems to be a long header packet when received by clients; but as the Version Field is 0, that packet is recognized as a *Version Negotiation Packet*. This packet is only sent by servers and is an indication of the server to a client which versions are supported by that QUIC server. All supported QUIC versions on the server are listed in that packet. It will be sent after the server received a packet with a version proposal that it does not support. If the client side chosen version is not supported by the server, the server has to send a *Version Negotiation Packet* which results in one additional round trip delay before the connection is established [11].

3.8. Multipath Extension

A further feature of the QUIC protocol is *Multipath*. A QUIC flow is, in contrast to a TCP connection, not bound to the 5-tuple consisting of source IP/Port, destination IP/Port, and the transport protocol itself. Instead, the protocol specifies a *Connection ID* each one for the server and the client which is placed in every QUIC packet header. Users can switch from one to another network seamlessly and still communicate with the same server. Using multiple paths over the Internet, with changed values in the 4-tuple, is specified through the QUIC Migration feature. A QUIC implementation including the *Multipath* extension in Go can be found in [3].

4. Related Work

HTTP/1.0 was standardized by the *IETF* in 1996 in RFC 1945. Three years later, the *IETF* introduced the second version HTTP/1.1 which is described in a stricter way with clearer rules in RFC 2616. Companies like Google and Microsoft always want the Internet to be faster. Google initiated the SPDY project [12] and on the basis of SPDY, the new version of HTTP was standardized by *IETF*: HTTP/2.0 was announced in 2015 (RFC 7540). It uses the multiplexing technology which transfers more data. On the incoming side, data is demultiplexed again. Among other reasons, and because of that, HTTP/2.0 is faster than HTTP/1.1. QUIC is the continuous development of Google's research to reduce latency in the web and transmit data faster. It is the successor of SPDY and the standardized HTTP/2.0 protocol.

The *IETF* currently has a working group for the transport protocol QUIC. Beside the core overview, each essential feature of QUIC is described into separate drafts which are planned to be standalone RFCs. There is also a working group responsible for "HTTP over QUIC" which describes the transport of HTTP over QUIC as transport layer. The researchers recently discussed about naming conventions and decided to rename "HTTP over QUIC" to "HTTP/3" [13]. HTTP version 3 would mean that the HTTP protocol would run on the base of the QUIC Transport Protocol. This would be another milestone for the future of the Internet.

To reduce handshake latency in TCP, there were some improvements. TCP Fast Open (TFO) allows to send data in the TCP SYN field to same servers. Thus, after connection establishment, there is a 0-RTT to send payload. But data can only be sent as much as the TCP SYN segment offers. QUIC does not have this limitation. Only the congestion controller or the flow controller can limit the data which can be sent in 0-RTT handshakes [1].

Another affiliated aspect in relation to QUIC is the Stream Control Transmission Protocol (SCTP) defined in RFC 4960. SCTP has many similarities and parallels to TCP. It is a connection oriented transport protocol which also uses sequence numbers and acknowledgments to provide reliable data delivery. Like TCP, SCTP uses a window mechanism to signal how much data can be delivered on receive buffers. Even if the terminology of SCTP is quite different, they both share common features. However, SCTP was built to introduce some advantages over TCP. It is not bound to a single IP address, not even on the IP versions. Moreover, both hosts of a SCTP connection can have multiple IP addresses to communicate with each other. SCTP transmits on multiple streams and is not bound on delivering data in order [14]. There is no head-of-line-blocking. The main problem of SCTP is that is not widely deployed. While the Linux kernel implemented SCTP, MacOS and Windows do not support SCTP officially, but only through extensions. Thus, SCTP is not spread widely. QUIC takes the main idea of SCTP and introduces stream-multiplexing without head-of-line-blocking. It provides *Multihoming* through the *Multipath* extension. However, through the aggregation of the transport and security protocols and the usage of UDP, handshake latency is reduced by QUIC.

5. Conclusion and Future Work

Latency Reduction was a key note when QUIC was developed by Google. Simultaneously, other transport features within the Internet such as security and reliability had to remain unchanged. Google reached these goals by deploying QUIC and tested it in production mode within Chrome/Chromium on YouTube and other Google services. The underlying UDP as transport layer is a new approach to deliver data still reliably. This leads to implement additionally separate features like congestion control, flow control, and loss recovery, but on another level and on top of UDP. That causes QUIC to run in user space. Indeed, features like latency improvements and removing head-of-line-blocking make QUIC attractive and let QUIC be a proposed standard for today's Internet.

doi: 10.2313/NET-2019-06-1_16

However, the protocol violates the OSI model. The transport features and the security protocol are mapped into the same layer in the application level without following the widely established OSI reference model. Thus, it remains to be seen which influence on the usage of the traditional TCP/TLS infrastructure will be. It also remains to be seen how the standardization process by *IETF* evolves and when the standard will be announced. Following the *IETF* milestones of the QUIC working group, most of the drafts' due dates were changed from November 2018 to July 2019 and one to May 2020. The *"Core Protocol document"* of QUIC is planned to be finished in July 2019 [6].

References

[1] A. Langley *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842

[2] D. J. Ledkov, J.-B. Lallement *et al.*, last visited 14 February 2019. [Online]. Available: https://wiki.ubuntu.com/CosmicCuttlefish/ReleaseNotes

[3] L. Clemente, M. Seemann *et al.*, "lucas-clemente/quic-go," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/lucas-clemente/quic-go

[4] D. Tikhonov *et al.*, "litespeedtech/lsquic-client," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/litespeedtech/lsquic-client

[5] B. Saunders, D. Ochtman *et al.*, "djc/quinn," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/djc/quinn

[6] "QUIC (quic)," last visited 17 February 2019. [Online]. Available: https://datatracker.ietf.org/wg/quic/about/

[7] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.

[8] "Playing with QUIC," last visited 17 February 2019. [Online]. Available: https://www.chromium.org/quic/playing-with-quic

[9] [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-recovery

[10] "SNMP counter," last visited 14 February 2019. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/snmp_counter.html

[11] "QUIC: A UDP-Based Multiplexed and Secure Transport." [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-transport-16

[12] "SPDY: An experimental protocol for a faster web," last visited 14 February 2019. [Online]. Available: https://dev.chromium.org/spdy/spdy-whitepaper

[13] M. Nottingham *et al.*, "IETF Mail Archive," last visited 14 February 2019. [Online]. Available: https://mailarchive.ietf.org/arch/msg/quic/RLRs4nB1lwFCZ_7k0iuz0ZBa35s

[14] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct 2006.

ISBN 978-3-937201-64-1
DOI 10.2313/NET-2019-06-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)