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 2019/2020

Munich, Germany

Editors

Georg Carle, Stephan Günther, Benedikt Jaeger

Publisher

Chair of Network Architectures and Services

Network Architectures and Services NET 2020-04-1

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

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

Winter Semester 2019/2020

Munich, August 2, 2019 – February 23, 2020

Editors: Georg Carle, Stephan Günther, Benedikt Jaeger

Network Architectures
and Services
NET 2020-04-1

Proceedings of the Seminar
Innovative Internet Technologies and Mobile Communications (IITM)
Winter Semester 2019/2020

Editors:

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

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

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

Cataloging-in-Publication Data

Seminar IITM WS 19/20
Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)
Munich, Germany, August 2, 2019 – February 23, 2020
ISBN: 978-3-937201-69-6

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2020-04-1
Innovative Internet Technologies and Mobile Communications (IITM) NET 2020-04-1
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 2019/2020. 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 Till Müller with the paper *File Injection for Virtual Machine Boot Mechanisms* and Stefan Waldhauser with the paper *Time Synchronization in Time-Sensitive Networking* .

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, April 2020

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

Sebastian Gallenmüller (gallenmu@net.in.tum.de)
Technical University of Munich

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

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

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

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

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

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

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

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

Jan Seeger (seeger@in.tum.de)
Technical University of Munich

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

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

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

Seminar Homepage

`https://net.in.tum.de/teaching/ws1920/seminars/`

Contents

Deep Learning on the mobile edge

Georg Eickelpasch, Marton Kajo*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email:georg.eickelpasch@tum.de, kajo@net.in.tum.de

***Abstract*—Applying Deep Learning on mobile devices proves to be a difficult challenge due to limited resources. However, it is still very much a required methodology, especially in the context of IoT devices, and therefore the computation is offloaded to the cloud or mobile edge. The offloading can be done dynamically were the device, edge and cloud share the work depending on different factors.**

***Index Terms*—Deep Learning, mobile edge computing, Internet of Things**

1. Introduction

This paper tries to give a comprehensive look at the questions of what applications benefit from using Deep Learning in a mobile context and how it can be applied. Because the data cannot be processed on the device, it has to be offloaded. This paper will take a look at different offloading strategies and when to use them.

Currently, Deep Learning is already used a lot on mobile devices, for example for speech recognition and image detection. However, a common approach is to offload the entire workload to the cloud. This brings many disadvantages, e.g. a strong dependency on a good bandwidth which can be a bottleneck. There are promising approaches, such as shown by Yiping Kang et al. in [1] to use the mobile edge as well as the cloud or completely replace the cloud by the edge like shown by En Li et al. in [2]. These algorithms offer a great improvement over cloud offloading but are still improvable. In this paper, we will show the strength and weaknesses of the algorithms as well as their common use cases.

Edge Computing is the idea to use locally existing hardware in the network instead of the cloud which is thought of a far away, more or less unlimited computing power. When talking about the edge, there are two different ways of thinking of the edge.

1) The first one is to think of the edge as the end-user mobile device with limited computing power and therefore tries to offload as much as possible to the cloud.
2) The second idea is that the edge is a powerful local server that offers the advantage of local caching and a low round trip time (RTT) for communication.

Even though, the ideas seem very different they can be used similarly. In both cases, there is one device with

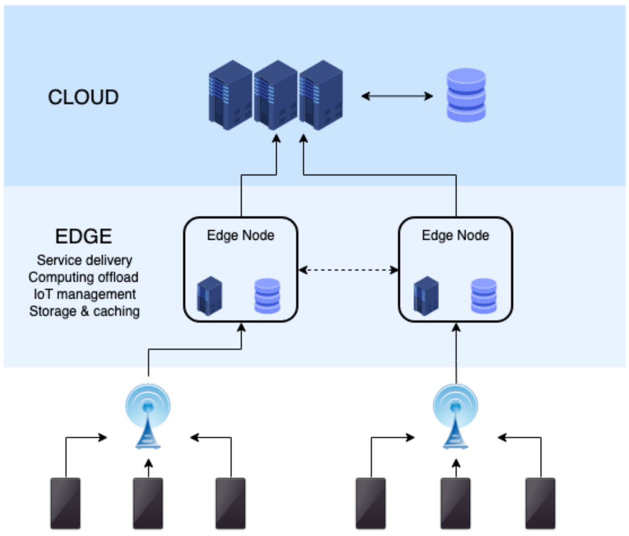


Figure 1: Different layers of Edge Computing Model [3]

weak computation power and one powerful device, which can be reached through offloading. Scheduling algorithms that offload between one computational strong and one computational weak component can be used in both cases, however, one has to be aware whether the edge is the strong component or the weak component.

2. Background

The offloading strategies discussed in this paper are based on the NeuroSurgeon paper [1] as well as the Edgent paper [2]. But while these two works focus on the details of the individual algorithm this work focuses on the high-level advantages and disadvantages of each. Furthermore, this paper shows the problems of Deep Learning on the mobile edge using the specific case of an IoT system as shown by Hakima Khelif et al. in [4] and He Li et al. in [5].
The usage of Deep Neural Networks (DNNs) is rapidly increasing in everyday life. Nowadays not only powerful computers but even mobile devices, e.g. smartphones use DNNs for various applications. Until recently, running a DNN on a mobile device seemed unfeasible. This is because mobile devices usually face the problem of limited resources. Restricted processing power and memory, as well as limited energy availability, meet the high computational effort of a DNN. But since the benefits and use cases of DNNs are more and more desirable the

scientific community is working on ways to execute DNNs on mobile devices. The easiest approach to offload the actual computing into the cloud, which works well but is bottlenecked by the data transfer between the mobile device and the cloud and therefore require high bandwidth for good performances. To increase this performance a hybrid model between the cloud and the mobile device with strategically and dynamically chosen offloading points is a viable attempt. But since the processing power on the device is highly limited it still leaves much room for improvement. The current state-of-the-art solution is another processing layer, between the mobile device and the cloud, called the Mobile Edge. Instead of offloading all heavy computational tasks to the cloud they are offloaded to strong processing devices nearby. This does not only improve the performance drastically and decreases the dependency on a strong internet connection, but also improves security, since personal data is not required to be offloaded to the cloud.

3. Use cases

In this section, we want to show which mobile applications require DNNs that run on the device or the edge.

3.1. Speech

The first thing that comes to mind is probably speech recognition. That means converting language spoken by a human into a machine-readable format. This problem used to be incredibly hard to solve and if solved rudimentary, is only able to detect very clearly spoken words or limited vocabulary which is not a sufficient solution for everyday applications like simple dictating tools for speech to text, since manually correcting errors can be very tedious. DNNs are suited well for speech recognition because they are more robust than alternatives and perform comparably well with noisy data. In combination with speech recognition, a second major field is often used because converting the speech to text is not always enough the machine also needs to understand what the user is speaking. For this Natural Language Processing (NLP) is required. Due to the complexity of natural languages, DNNs are a viable solution approach for NLP. However, NLP applications like intelligent personal assistants, e.g. Amazon Echo or Google Home require real-time processing since it is supposed to feel natural, like speaking to a human.

3.2. Computer Vision

Another problem commonly faced by mobile devices that requires DNNs is object identification on pictures. This could be a smart fridge checking what groceries need to be bought or a phone identifying the user via FaceID to unlock a smartphone. While the smartphone again faces the challenge of real-time results it has the new problem of a highly mobile context which leads to inconsistent bandwidth in some cases, so offloading to the cloud is unreliable. The smart fridge might not require real-time results but even though it is expected to be connected to a wifi network, bandwidth is a limited resource. This is due to the expected growth of the IoT and connected home devices. If all of these devices use bandwidth recklessly, it will lead to congestion in the network. Pairing Edge Computing with IoT devices can greatly reduce required bandwidth and therefore is a desirable methodology. A more advanced field in computer vision that requires DNNs is Continuous Vision. Tracking objects in a video in real-time is very useful on the one hand, but on the other hand, it is very challenging to do. Currently mostly used in robotics and surveillance it is expected to expand more into the private sector for example for video games. Being one of the computationally most expensive subtopics due to the sheer size of video files offloading everything to the cloud is not possible for real-time applications. Therefore preprocessing on the edge is currently the most promising approach.

3.3. Bioinformatics

Lastly, bioinformatics and especially E-Health is a fast-growing sector that requires DNNs due to noisy real-world data. To get good results with noisy data DNNs are a common approach. Also, E-Health requires very robust solutions since human lives might depend on it. Therefore, DNNs are the first choice. In the context of IoT devices offloading to the edge can also provide the advantage of anonymizing the data which is also especially important for human health data.

4. Deep Learning on the mobile Edge in an IoT context

In the previous section, many use cases for DNNs on the mobile edge were shown. Many of them were related to or applicable in IoT devices. In this section, we want to take an in-depth look at the problems IoT devices face and why DNNs are a good way to solve them. After that, we will add in Edge Computing as a solution to apply DNNs to IoT devices and see what other benefits arise.

4.1. DNNs in IoT

The IoT is a growing phenomenon where more and more devices are interconnected with each other and the Internet. Not only smart home solutions for private people but more importantly in factories are more and more robots and machines part of the IoT and it is expected to change, or already changing the way how pipelines work, in Germany under the synonym Industry 4.0 [6]. All of these devices use various sensors and communication to adapt in real-time to changing conditions. However, real-world data are always noisy or hard to predict. To work with this data DNNs are a powerful solution due to their robustness and the availability of a large amount of data. Of course not every IoT device has powerful computing capabilities so the actual DNN has to be computed externally.

4.2. Edge Computing for DNNs in IoT

Offloading the computation process traditionally means to upload the raw data into the cloud where everything will be fully computed and the result returned

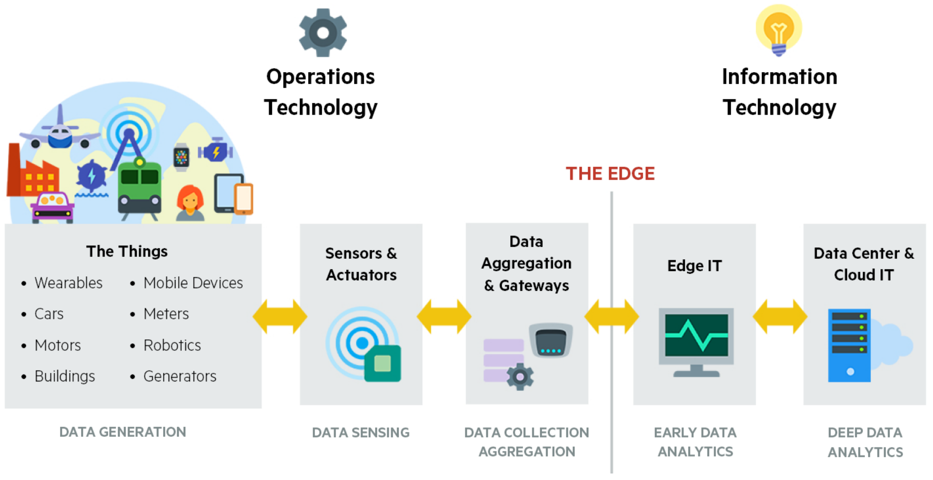


Figure 2: Workflow of an IoT-Edge solution [7]

to the IoT device. However, with more and more devices being part of the IoT uploading all the raw data themselves can already be too expensive or not be achieved within a required timeframe. Upgrading bandwidth and computational infrastructure are possible, but it also has limitations and can be very expensive, therefore a method to relieve the bottleneck which is uploading is very useful. This method is Edge Computing. Edge Computing can be done on the IoT device itself or a designated physically close edge server. The basic idea is to exploit the property of many DNNs that the input data are bigger than the intermediate data [2]. This means that the relevant data are smaller than the raw data, after processing the first layers. So the beginning of the processing is done on the edge device even though it computes slower than the cloud but the lost time is saved later when smaller data are uploaded to the cloud instead. By doing this the bottleneck is relieved of some workload. The question remains when exactly to offload to the cloud service. We will take a look at this question in the next section. But there are more benefits in using Edge Computing than just compressing the data by preprocessing them. Another typical feature of data acquired by IoT devices is the very high heterogeneity due to many different sensors and different use cases or environments. If everything is uploaded to the cloud this heterogeneity requires the cloud to be able to process many kinds of data. If Edge Computing is used the edge device can preprocess the data and since each edge device usually faces the same data from the same sensors it can be optimized to specifically preprocess this data and upload a more generalized dataset to the cloud. This localization of data preprocessing to the edge also enables strong caching mechanisms since the data of each sensor are processed locally every time - on the cloud, this would not be possible due to different sensor environments. By generalizing the data during the preprocessing on the edge it is not only faster computable by the cloud but it might also be very hard to reconstruct the original data because backtracking the layers of a DNN is a difficult problem. This is another advantage because it can be used to protect sensitive user data and the cloud never has access to the original data. Finally, by distributing the system onto many edge devices a certain layer of redundancy is added. In case of failure of an edge device only a small part of the network will not work properly, which might even be covered by other edge devices. This makes the system more resistant to typical attacks like DDoS because DDoSing an edge device would be useless and the cloud is less important to the entire system. Strengthening this single point of failure is especially an important feature for companies.

5. Scheduling of Offloading

In this section, we want to take a look at two recent scheduling strategies for edge assisted computing of DNNs. We want to clarify what variables exist in the offloading process and how they affect the offloading decision.

5.1. Preprocessing Strategy

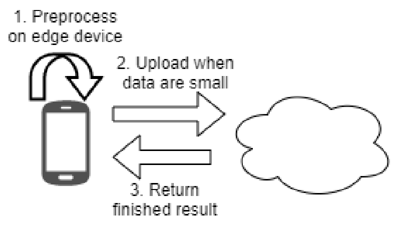
The general idea of this strategy is that the intermediate data will be smaller than the raw data. Therefore the data are first processed on the edge device and at a strategically chosen point uploaded to the cloud which does all the remaining computation. The offloading point is chosen based on the following factors. The first variable to consider is the architecture of the DNN. There is an in-depth analysis of layers in state-of-the-art DNNs that sets computational effort and size of output data into relation [1]. For the offloading decision, it is important that as much computational effort as possible is after the offloading as well as offloading after a layer where the output data is small. The next variable is the edge hardware. To accurately predict when to offload it is important to know how much computing power is available on the edge device. If an edge server is used this power may vary too based on current workload. The more computing power is available on the edge, the faster the edge device can compute deeper layers of the DNN where a smaller data output size might reduce uploading time and save bandwidth workload. The next variable is the network bandwidth. Especially on mobile devices like smartphones, the available network can vary strongly between wifi or mobile data. The slower the upload speed of the edge device is the more priority has to be put on uploading at a point where the data are small, while a fast upload might upload at a bigger point but therefore has to execute less computation on the edge device. Finally, the computational power of the cloud also impacts the decision. While the cloud is expected to have a much faster computation speed than the edge device, the speed of the cloud might be limited due to workload at peak times. After considering all these factors the decision when to offload can be made quite accurately and improve the execution speed of DNNs on mobile devices drastically. Furthermore, the same algorithm can be used to optimize the energy usage of the phone by adding consumed energy as a variable that can be traded for speed. If multiple devices run this scheduler are interconnected they could also balance the workload of the cloud intelligently.

5.2. Deadline Strategy

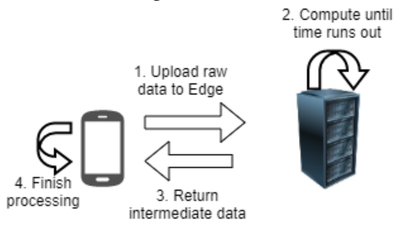
This general idea of this strategy is to trade off accuracy for time. While the previous strategy tried to optimize the time needed for the execution, this strategy expects to have a deadline until when it has to be finished. This can be useful in real-time contexts where the results have to be delivered on time, e.g. for autonomous driving danger has to be spotted at a time when the car is still able to react. If the calculation would be completed later, the result would be of no use. Therefore accuracy is a new variable. To have a variable accuracy a DNN with multiple exit points is used. Unlike the first strategy, the deadline strategy always uploads the raw data but it does not receive the final result instead receives an intermediate value and will finish the computation on the device. By early-exiting the DNN it can be guaranteed that a deadline can be met while the accuracy is as high as possible for the given timeframe. However, to be able to accurately predict deadlines other variables have to be more stable or even fixed. The Edgent algorithm [2]] assumes a model where the data is not processed on the cloud but instead on a very powerful edge device. That offers the benefit of strong and consistent bandwidth. While DNN architecture and the used hardware are still variable for the application they are fixed after the first setup, leaving as little variables as possible to guarantee performance.

6. Conclusion and future work

Deep Learning is an important state-of-the-art methodology with many use cases. Especially with the growth and integration of IoT into everyday life, it is expected to stay very relevant in the future. To make this methodology available for all devices and real-time applications it has to be offloaded since processing DNNs on small or weak devices would take too long or trades off accuracy [8]. Instead of loading everything into the cloud, using Edge Computing offers great advantages. Optimizing the offloading process is important due to the expensiveness of DNNs and there is a lot of potential for research in that area. Future works could be about optimizing Edgent with the NeuroSurgeon baseline since Edgent currently is not optimized in that regard. In the context of IoT, a new algorithm with two offloading stages could be considered, where one stage is computed on the IoT device, one stage on an edge server and one stage in the cloud. This would offer very dynamic computing capabilities, high speed for regular workload due to a strong edge server and strong robustness in case of peak workload due to the cloud.


1. Preprocess on edge device
2. Upload when data are small
3. Return finished result

(a) Neurosurgeon's basic workflow


2. Compute until time runs out
1. Upload raw data to Edge
4. Finish processing
3. Return intermediate data

(b) Edgents basic workflow

Figure 3: Comparison of the two workflows

References

[1] Y. Kang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," 2017.

[2] E. Li, "Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy," 2018.

[3] "Edge Computing Model," https://upload.wikimedia.org/wikipedia/commons/b/bf/Edge_computing_infrastructure.png, [Online; accessed 29-September-2019].

[4] H. Khelifi, "Bringing Deep Learning at the Edge of Information-Centric Internet of Things," 2019.

[5] H. Li, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," 2018.

[6] C. Towers-Clark, "Big Data, AI & IoT Part Two: Driving Industry 4.0 One Step At A Time," 2019, [Online; accessed 29-September-2019].

[7] E. Signorini, "HPE and IoT Compute at the Edge," 2016.

[8] Y. Li and T. Strohmer, "What Happens on the Edge, Stays on the Edge: Toward Compressive Deep Learning," 2019.

File Injection for Virtual Machine Boot Mechanisms

Till Müller, Johannes Naab*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: till.mueller@tum.de, naab@net.in.tum.de

***Abstract*—Virtual machines are widely used in today's computing infrastructure. They provide an easy way to isolate untrusted software and users from each other and the hosts they run on. This isolation, however, makes it difficult for administrators to exercise control over the VMs without compromising security.**

We provide a system that allows the admin to inject files into untrusted VMs in a secure manner. It uses a customized Linux kernel which is booted using QEMU direct kernel boot. To inject files safely, they are passed to the virtual machine via the `initramfs`, a read-only archive normally used to provide drivers to the kernel during boot. The file injector then uses `kexec` to load a kernel from the guest's filesystem, thereby assuming the functionality of a bootloader like GRUB to minimize user impact.

***Index Terms*—virtual machines, qemu, kvm, file injection, Linux kernel boot**

1. Motivation

The machines used in this project are untrusted virtual machines with potentially malicious actors having access as `root`. Nevertheless, these VMs need to be administered (i.e. to grant access or run updates), a non-trivial task to accomplish while not compromising security since most administration tools assume they have direct access to the machine that is being administrated. To protect the VMs' host, the file injector should not need to run any code originating from the VMs and overall enforce strict isolation.

This project has been motivated by the lecture "Grundlagen Rechnernetze und verteilte Systeme" [1] which requires such a tool to provide virtual machines for students taking the lecture.

For this lecture, every student is given access to a virtual machine to use for developing and debugging of homework assignments. The students log in as `root` on their respective machines, giving them full control over their VM. It is therefore paramount to keep these VMs isolated from the host system. All requirements and assumptions are based on this scenario.

This paper is structured as follows: In Section 2 we analyze the background of the technologies we used, as well as existing boot processes. The 3. Section evaluates different possible solutions to the issues described here. In Section 4 we present our bootloader and file injector and in Section 5 the performance of some boot methods is highlighted. Section 6 concludes the paper.

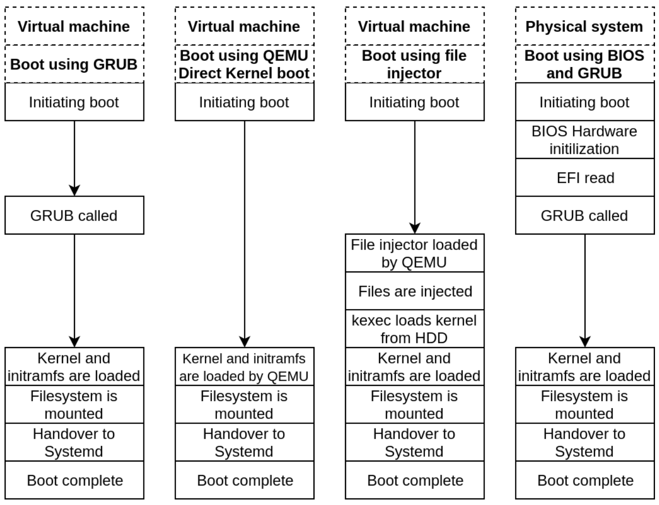


Figure 1: Flowchart of boot processes

1.1. Requirements and limitations

For our boot loader and file injection system we propose the following requirements:

- External data has to be written to the VMs' filesystem
- The VMs have to be persistent between reboots
- Boot times have to stay reasonable
- The system has to be able to scale to hundreds of machines
- Users should be able to update their VMs' kernel independently from the file injector
- The setup must not require a network connection to inject the files or boot the VMs
- In case the boot process fails, manual debugging should be possible

1.2. Assumptions

To make the implementation easier, the file injector is based on these assumptions:

- Virtual machines are hosted using QEMU/KVM with libvirt
- The VMs run Debian and are all set up in a similar way
- Only small configuration files need to be copied to the guests' filesystem

2. Background

In this section we will take a closer look at how a virtual machine boots using different methods. During this, we will also evaluate and explain how these technologies might be useful as part of the file injection system.

2.1. GRUB

A conventional boot loader like GRUB [2] or LILO gets invoked by the BIOS/EFI at the beginning of the machine's boot process. It then lets the user choose which operating system to boot. For this, GRUB reads the partition table, which contains information about what partitions are bootable and where on the disk they are located. Afterwards, GRUB loads the kernel from the chosen partition and hands over to it. The kernel then starts its first process, on Linux systems usually init or Systemd.

2.2. QEMU direct kernel boot

A different way of booting a QEMU-hosted virtual machine is the so-called direct kernel boot. In normal use, this enables a VM to skip the bootloader by using a kernel file from the host system. The virtual machine then jumps directly to the init or Systemd invocation, thereby not requiring a kernel to be present on the VMs HDD and speeding up boot-times. In the case of injecting files however, the direct kernel boot feature is used to boot up the file injector without the need to use the VMs' filesystem as intermediary storage, making the system less error-prone and resulting in no unwanted side-effects during this first boot stage (Fig. 1). This is possible because no part of a direct kernel boot requires the HDD, which is only mounted at the end of the init process.

2.3. initramfs

When booting a Linux kernel of almost any kind, two files are involved: The first is the Linux kernel itself, which contains the kernel code, the second is the *initial ram filesystem*. The `initramfs` contains all drivers and software needed to mount the filesystem from the HDD, e.g. RAID or decryption tools. It is a `cpio` archive which the kernel uncompresses into a temporary filesystem (TMPFS) during boot. This filesystem then resides entirely in the machine's memory, enabling the kernel to load modules required to continue booting.

The old format for this file was `initrd`, which behaved similar to a block device using a filesystem like `ext2`. Today, this format is rarely used, although the name regularly appears in commands, e.g. for QEMU direct kernel boot, even though an `initramfs` is used.

2.4. kexec

`kexec` is a program that emulates the function of a bootloader from within a running system. When running `kexec`, it initiates a normal shutdown, but immediately restarts the system using the chosen kernel. Therefore, no shutdown signal is sent to the motherboard and BIOS; hardware initialization and the bootloader are skipped. `kexec` is therefore similar to QEMU direct kernel boot: Both start the system with the kernel immediately available. In our implementation, `kexec` is used to load a user-provided kernel after the files have been injected.

3. Alternative approaches

Before the final decision on the implementation was made, other possible approaches to the problem had to be evaluated as well. We will, therefore, take a look at using existing systems, such as Virtio's `virt-install` or PXE.

3.1. Using direct kernel boot

The original setup we based this work on was similar to the one that was eventually chosen, with one major difference: Instead of using `kexec` to find and boot a kernel from the machine's filesystem, the injector would continue booting normally and therefore be the active kernel while the machine was running. This enabled a quick boot process (see benchmarks in Section 5).

The downside to this approach was that updating the kernel came with a lot of issues. The main one was that the kernel version installed inside the virtual machine and the one the machine was booted with had to be kept in sync.

This was required because the kernel loads version-specific modules from the machine's filesystem after it has been mounted. To load these modules, the kernel expects a folder named after the kernel version in `/lib/modules`. If this folder does not exist, the modules are not loaded. As a result, updating the kernel the machines were booted with was not an option since it would have led to all machines losing the functionality these modules provided (e.g. ACPI support).

Updating the kernel within the virtual machine did not have any effect due to the kernel being loaded during boot still being the same one originating from the host system. This could lead to user frustration, especially when building kernel modules, and while the manual usage of `kexec` could circumvent this limitation, a system is preferred that does not require such a workaround from the users.

3.2. virt-install

Virtio's `virt-install` is a tool to set up new virtual machines using a predefined image. When given a kickstart file, `virt-install` can make configuration changes and copy data to the machine.

While `virt-install` can import existing images, we were unable to find a way to for `virt-install` to alter the contents of the VMs disk image during this process. `virt-install` can edit a virtual machine's `virsh` configuration file, but this only allows it to change, for example, the connected devices or allocated resources to the VM. It was therefore ruled out after some initial testing in favor of the other ideas described here.

3.3. Mounting the guest's filesystem on the host

This approach was deemed too slow and insecure to use for untrusted VMs. While there should be no way for a mounted filesystem to execute any code, keeping the filesystems separated is the best way of ensuring isolation. Modern filesystem like `ext4` trust the images they mount, so a filesystem image crafted with malicious intentions could cause major security issues [3]. One solution here is using `libguestfs` [4], which mounts the guest's filesystem inside a minimal VM and therefore enables the host to securely alter the guest's HDD contents. This method, however, is unsuitable for our purpose, since the process would increase boot times significantly. Additionally, mounting and unmounting a filesystem with this method every time a VM boots can put additional load on the host, especially if multiple virtual machines are booted at the same time.

3.4. Network boot / PXE

The Preboot Execution Environment (PXE) is a standardized process of booting a physical or virtual machine over the network using a TFTP (trivial file transfer protocol) server. The PXE implementation by Syslinux [5] can transfer a kernel and `initramfs` file to the booting machine. Unfortunately, PXE requires a network connection and DHCP to connect to a server. Additionally, the `pxelinux` implementation does not provide tools for running scripts on boot.

After considering these issues, PXE was ruled out as a solution as well. While it might be possible to achieve the desired outcome using it, the infrastructure defined in the requirements does not provide DHCP, making PXE impossible to use.

3.5. Ansible

Another tool often used for applying configurations on multiple machines is Ansible [6]. It can automatically provision and set up physical and virtual machines, all configured using YAML files as so-called Playbooks. Like PXE however, Ansible needs a connection to its server to download these Playbooks, which makes it unsuitable for the requirements described in Section 1. Ansible also runs as an agent on the machines it manages, which would enable users to disable it, rendering it useless.

4. Architecture

All approaches listed in the previous section either do not fulfill some of the requirements or are not compatible with the limitations. The system described below was built upon some of them to achieve all desired results within the limits set in Section 1.

In summary, the file injector runs on the virtual machine before its real kernel is booted from the HDD, enabling full access while being controlled by the administrator.

4.1. Replacing the bootloader

Since the code to inject files needs to run before handing over control to the untrusted guest system, the injection process takes the place of the bootloader. QEMU direct kernel boot loads the injector and runs it inside the virtual machine. This behavior is not possible when using GRUB, since the injector and the files to inject are not located on the VM, but on the host's filesystem.

4.2. Injecting files

The file injector works in two stages:

1) Mount the machine's filesystem and inject the files
2) Find the correct kernel to switch to and execute it using `kexec`

During the first stage, the injector needs to find the `root` filesystem and mount it. Like a kernel during normal boot procedures, `mount` needs to know the label, UUID or path to the block device containing that filesystem. The type of the filesystem used is also required, although `mount` is sometimes able to get it from `blkid` or the `volume_id` library. Normally, `mount` would also try to read `/etc/filesystems` and `/proc/filesystems`, but these are not available in the `initramfs` environment [7]. The default values for both the block device's location and its filesystem type are set in a way that makes them compatible to most virtual machine setups. Additionally, it is important here that the guest's filesystem is mounted with read/write options. While `initramfs` normally mounts the filesystem as read-only, this would not be sufficient for file injection.

To enable injection, `initramfs` delivers the files the injector needs to copy to the guest's filesystem. While this limits the size of the files the system can inject, using `initramfs` is a fast, reliable and secure way to move files from the host's filesystem to the guest's.

4.3. Booting the correct kernel

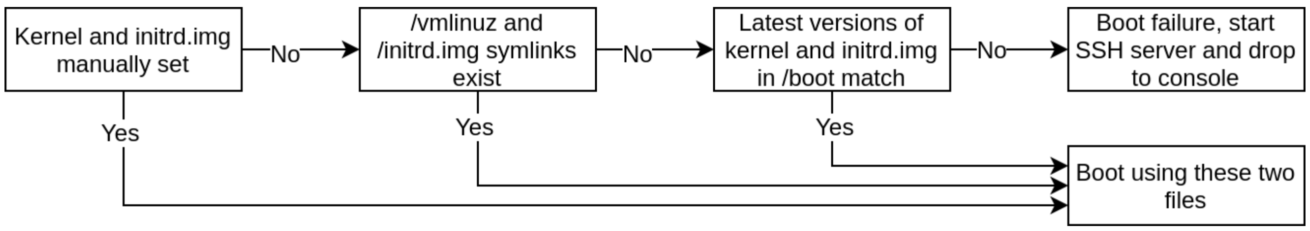
After the first stage, the injector needs to load the right kernel to properly boot the system. During the second stage, the injector looks for a suitable kernel/`initramfs` combination to switch to. There are two possible locations for these files: They are either linked from `/vmlinuz` and `/initrd.img` or are located in `/boot`. The administrator can also set both manually using the boot parameters, as shown in Figure 2.

After the injector has found the kernel and `initrd.img`, it loads them using `kexec`, unmounts the filesystem and `kexec` switches to that kernel which continues the boot process.

If the system is unable to find a working kernel/`initrd.img` combination or `kexec` fails, it drops to a shell and automatically tries to start an SSH server so the issue can be debugged and resolved manually.

4.4. Implementation

Implementing the system as described above has produced several challenges that needed to be solved. Some of them are described below, along with possible solutions.



Kernel and initrd.img manually set
No
/vmlinuz and /initrd.img symlinks exist
No
Latest versions of kernel and initrd.img in /boot match
No
Boot failure, start SSH server and drop to console
Yes
Yes
Yes
Boot using these two files

Figure 2: Flowchart of how the kernel / initrd.img are chosen

4.4.1. Passing files to the injector. The main issue resulted from a combination of the requirements and limitations imposed by QEMU and its direct kernel boot-feature. To pass files to the injector and therefore the guest without a network connection or shared filesystem, they must be included in the `initramfs`. Creating a `cpio` archive and setting it as the `initramfs` for the QEMU direct kernel boot, however, is not sufficient. This is caused by the issue mentioned before (see Section 2.3) that the kernel itself does not contain all drivers to boot the system and requires an `initramfs` with additional modules.

This means that a customized `cpio` archive needs to be available for each virtual machine during the boot process. `cpio` archives can be concatenated, so simply combining the archives containing the files and drivers would result in one which contains all required files. This however creates another issue: The newly created archive would not only contain the files the injector needs to inject, but also about 27MB of static data required by the kernel.

Generating these archives on every boot for hundreds of machines is wasting time and space, so we focused on finding a way to circumvent this issue.

4.4.2. Including the static initramfs in the kernel file. Since most of the `initramfs` file is static, a possible solution is to include it with the other static file used during boot, namely the kernel. This solution proved to be the best one because it enables the `initramfs` given to QEMU to just include the files the system injects. When compiling the Linux kernel, one available option is `CONFIG_INITRAMFS_SOURCE`. The kernels shipped by Debian do not use this option, resulting in the kernel containing an empty `cpio` archive. Being able to include the static `initramfs` in the kernel though allows the injector to be one self-contained kernel file which includes everything needed to boot the system. The only downside to this approach is the added complexity from having to recompile the kernel to make changes to the file injector.

4.4.3. Adding the initramfs to a precompiled kernel. In theory, it is also possible to achieve the results from the previous section without having to recompile the kernel by editing the Linux kernel file (`vmlinuz`) to include a different `initramfs` from the one it was compiled with. However, making this work requires changes to multiple values within the kernel file [8] [9]. This means that not only the data has to be in the right place, but offsets and some pointers have to be updated as well. Therefore, this process was deemed too fragile to achieve the desired result.

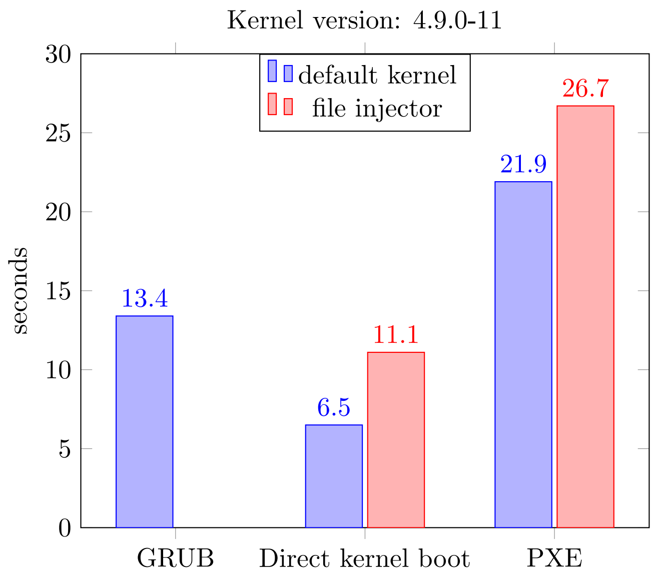


Figure 3: Boot times using an unaltered Debian kernel and the file injector

4.5. Executing the injector script

The `initramfs` that includes the file injector is based on a Debian `initramfs`, therefore requiring changes to adapt it for the injector. For example, some effort was needed to find the best way of getting the `init` process to execute the file injector script at the right time. Since `init` normally executes all scripts in the `/scripts/init-{premount,top,bottom}` folders as defined in the `ORDER` files placed in them, the init process was still executing the mounting mechanism normally required during boot. This could break the mount attempted by the injector, so the behavior had to be changed.

To accomplish this and to have more control over the `init` process, the `init` script was altered in the filesystem overlay to switch to the injector as soon as possible, effectively stopping at that point and making sure that it does not interfere with the file injection.

5. Benchmarks

A quick overall boot process was one of the goals while implementing this system. Therefore, we now take a look at how much longer booting takes while using the file injector. These tests were conducted using the same VM and the times were measured from sending the start command via `virsh` until the login screen was displayed. The results are shown in Figure 3.

Having GRUB boot an external file is outside its scope, so only the unchanged Debian kernel from the filesystem image was tested. Furthermore, the GRUB configuration was not altered, which resulted in the default five seconds delay during which the user can choose another operating system. Since this delay would exist on newly installed VMs as well, it was left in.

Even though the Preboot Execution environment was already ruled out in Section 3, we have included it here for comparison. It is immediately obvious that booting with `pxelinux` takes longer due to the need for a network connection. In this case, the DHCP- and PXE servers were located on the same host, so network latency or bandwidth limits are unlikely to have influenced these results.

The file injector increases boot times by about five seconds. The tests have been performed without any files to inject, so if the injector needs to copy large files, this would increase the boot time further. Copying large files, however, is not the intended use case for this implementation, we mainly focused on small configuration files. Booting a kernel directly is the fastest, but with the file injector using direct kernel boot being similar in boot time to using GRUB without file injection, the impact on users is negligible.

6. Conclusion

We developed a `kexec`-based boot loader for virtual machines. It is based on a modified Debian `initramfs`, which is directly embedded into a rebuilt kernel image. It allows to manage VMs by injecting configuration files and does not require mounting the guest's filesystem on the host. The file injector works reliably and will enable administrators to easily configure the untrusted systems hosted on their infrastructure.

Future work includes making the system more compatible to non-Debian distributions and adding functionality like deleting files in addition to injecting them, as well as passing arbitrary parameters to the user-kernel's command line.

References

[1] "Lecture 'Grundlagen Rechnernetze und verteilte Systeme'," https://www.net.in.tum.de/teaching/ss19/grnvs.html, [Online, accessed 19-September-2019].

[2] "GNU GRUB," https://www.gnu.org/software/grub/, [Online, accessed 26-September-2019].

[3] "On-disk format robustness requirements for new filesystems," https://lwn.net/Articles/796687/, [Online, accessed 25-September-2019].

[4] "libguestfs," http://libguestfs.org/, [Online, accessed 25-September-2019].

[5] "PXELINUX, official site," https://wiki.syslinux.org/wiki/index.php?title=PXELINUX, [Online, accessed 19-September-2019].

[6] "Ansible," https://www.ansible.com/, [Online, accessed 27-September-2019].

[7] "mount(8) - Linux man page," https://linux.die.net/man/8/mount, [Online, accessed 19-September-2019].

[8] "Stackexchange: Replacing section inside elf file," https://reverseengineering.stackexchange.com/questions/14607/replace-section-inside-elf-file, [Online, accessed 19-September-2019].

[9] "Gentoo forums: Linking existing kernel with new initramfs," https://forums.gentoo.org/viewtopic-t-1087792-start-0.html, [Online, accessed 19-September-2019].

Natural Evolution Strategies for Task Allocation

Emir Besic, Jan Seeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: emir.besic@tum.de, seeger@in.tum.de

***Abstract*—In this paper we will be taking a look at Natural Evolution Strategies as a possible solution to the task allocation problem. Task Allocation appears in many areas of science, but is still missing a perfect solution. As the problem is NP-hard, it isn't possible to find a fast algorithm that always provides the optimal solution, which is why it is mostly solved using heuristic approaches. Natural Evolution Strategies is one such heuristic approach, showing promising results in the function optimization area. We take a look at the theory behind using it and compare it to different approaches, which are currently being used.**

***Index Terms*—task allocation, natural evolution strategies**

1. Introduction

Task allocation is a problem which occurs in many different forms, but the general form can be described as follows: There are tasks which need to be done, as well as agents which can do those tasks. For an agent to do a specific task he needs to pay a specific cost and he gets a specific profit. Each agent has a limited budget and he can only do as many tasks as he can pay for. The goal is to allocate the tasks to the agents, such that the profit is maximized and every agent stays within their budget.

One example is the factory allocation issue, which can be defined as follows. You can build a set amount of factories and need to allocate them to different locations. Each location has differing costs and profits associated with it. The factories would be the tasks and the locations would be the agents in this case. You can only place 1 factory at a location, so the budget of the agents would be 1 and each location offers a different profit due to a variety of reasons like tax and surroundings.

It is a very relevant problem lately, due to the rising popularity of distributed systems and IoT. Task allocation is a very common problem in these areas, so there is a need for good solutions.

There are some approaches which are still used and guarantee an optimal allocation, most prominent of which being the branch-and-bound algorithm. Jensen et al. in [1] describes the basics of this approach. They enumerate the candidate solutions in a special way, which enables them to search parts of the solution space only implicitly. In other words, it is possible to eliminate multiple candidate solutions just by checking one.

Unfortunately, finding the optimal allocation is NP-hard as proved by Cardellini et al. in [2], which means that it is not possible to find it in polynomial time. For this reason the problem is most commonly solved with heuristics. Heuristics may return sub-optimal solutions, but they are much faster than traditional approaches like the branch-and-bound solver.

There are many different heuristics which may be used for task allocation and we will be describing some of them in section 2. In this paper, we will be looking at a heuristic which shows a lot of promise, Natural Evolution Strategies (NES). NES is a state of the art function optimization algorithm which was first mentioned by Wiestra et al. in [3]. We will be describing NES in more detail in section 4. It isn't possible to directly apply NES for solving Task Allocation, which is why it needs to be adapted for this use-case. We will be taking a deeper look at why it is not suitable and a possible solution in section 5.

Another aspect is that there may be some optimization goals, which are specific to that instance of the problem. These goals tell the algorithm which solutions to prioritize. One such goal is optimizing energy usage as Seeger et al. describes in [4], where an allocation is considered optimal if it minimizes total energy use over an execution. Another optimization goal is response time and availability as Cardellini et al. describes in [2]. More optimization goals would be to minimize network usage, end-to-end latency, inter-node traffic etc. We will describe some use-cases in detail in section 2.

2. Background and Related Work

In this section we will be looking at some use-cases of task allocation.

Cardellini et al. considers the optimal operator placement for distributed stream processing applications in [2]. This is a basic use-case of the Task Allocation Problem. She has formulated the problem and implemented a solution which can be used as a benchmark against which to compare other placement algorithms.

Stanoi et al. looked at the problem of the distribution of query operators over a network of processors in [5] . He adapted a hill-climbing heuristics to reduce the search space of configurations.

Rizou et al. developed a distributed placement algorithm that minimizes bandwith-delay product of data streams between operators. He used a heuristic that first calculates the solution in an intermediate continuous search space and then mapping it to the physical network. ([6])

Gerkey et al. in [7] considers multi-robot task allocation (MRTA), which is also one of the textbook examples of our problem. He has formulated MRTA in a formal manner and given some commonly-employed and greedy solutions for the easier problems.

Seeger et al. tackles the problem of a malfunctioning IoT device, by allocating its tasks to other devices in the network in [4]. He solves the allocation problem by removing some constraints to transform it into a linear problem and using the simplex method [8].

In [9], Lewis et al. tackles the general problem by re-casting it into the form of an unconstrained quadratic binary program (UQP), which is then solved by a tabu search method developed for solving the general UQP model.

Cano et al. considers the problem of assigning software processes to hardware processors in distributed robotics environments in [10]. They model it as a task allocation problem and use constraint programming, a constructive greedy heuristic and a local search meta-heuristic to solve it.

In [11], Wun-Hwa Chen et al. considers a special form of task allocation where they attempt to assign tasks to processors such that the communications cost among the processors, as well as the fixed costs of the processors are minimized. To solve this they use a hybrid method that combines Tabu search, described by Glover et al. in [12], for finding local optimal solutions and noising methods, described by Charon et al. in [13], for diversifying the search scheme.

Using evolution strategies for solving Task Allocation is not a new Idea, as it has already been discussed by Gupta et al. in [14]. The key difference is that they only consider some basic evolution strategies. But they already get some promising results, which gives us hope that NES might perform even better.

There have been innumerable other examples of Task Allocation, all with slightly different solutions. But it should be apparent now that this is a common problem without a commonly accepted perfect solution.

3. Modeling the Task Allocation Problem

We will base the model on the approach from Cardellini et al. in [2]. Let A denote the set of Agents and T the set of Tasks. Furthermore let P be a matrix such that the element $p_{\mathrm{i,j}}$ represents the profit when agent i does task j, let C be a matrix such that the element $c_{\mathrm{i,j}}$ represents the cost of agent i doing task j and let B be a set which contains the budget information of each agent such that agent i has budget b_{i}. There is another overview of the described parameters in table 1.

Symbol	Description
$\mathrm{a_i}$	Agent with the index i
$\mathrm{t_i}$	Task with the index i
$\mathrm{p_{i,j}}$	Profit when agent i does task j
$\mathrm{c_{i,j}}$	Cost when agent i does task j
$\mathrm{b_i}$	Budget of agent i
$\mathrm{x_{i,j}}$	Represents if agent i was assigned task j

(1)

With these definitions we can now define Solving the Task Allocation problem for n tasks and m agents as the process of maximizing

$$\sum_{i=0}^{m}\sum_{j=0}^{n} p_{\mathrm{i,j}}x_{\mathrm{i,j}} \tag{2}$$

while also staying withing the budget for each i:

$$\sum_{j=0}^{n} c_{\mathrm{i,j}}x_{\mathrm{i,j}} \leq b_{\mathrm{i}} \tag{3}$$

Where $x_{\mathrm{i,j}} = 1$ when task j was allocated to agent i and 0 otherwise. Another constraint is that a task can only be assigned to a single agent which means that

$$\sum_{i=0}^{m} x_{\mathrm{i,j}} = 1 \tag{4}$$

for each j. In most instances of this problem there would be more constraints, but for the sake of simplicity, these will be the only constraints we will consider as they appear in every instance of the problem. This is a common formulation of the problem and it can also be used with a branch-and-bound solver in order to find the optimal allocation.

4. Natural Evolution Strategies

Function optimization problems appear in a variety of different scientific areas. Often these problems are not feasibly solvable in a short amount of time. Thankfully, small errors can sometimes be tolerated and the solution does not need to be optimal. This is where a heuristic approach like Natural Evolution Strategies (NES) ([3] and [15]) comes into play. They can find a solution to the problem in a much shorter time. The solution they come up with however, may not always be optimal, which is why it's important to pick the right heuristic. NES is one such algorithm which uses an heuristic approach for performing 'black box' function optimization. The structure of this function, also known as the fitness function, is unknown, but some measurements, chosen by the algorithm, are available.

4.1. How NES Works

To understand how NES functions, we need to first understand how the basic Evolution Strategies (ES) work. They are named as such due to their inspiration from natural Darwinian evolution. The basic idea is to produce consecutive generations of samples (candidate solutions) with each generation coming closer to the optimal result. We initialize the algorithm with a set of samples. They are then evaluated using the fitness function. The ones with the best performance are then chosen to create the next generation by *mutating* their genes, while the others are discarded. The Process is continued until a satisfying result is reached. This approach was proven to be very powerful, but it does have many problems. The most prominent being the high sensitivity to local optima (sub-optimal solutions) and the slow speed of the evolution.

The Covariance Matrix Adaptation (CMA) algorithm is a much more sophisticated evolution strategy. CMA does not discard bad samples, but uses them to generate correlated mutations, which substantially speeds up evolution. It uses a multivariate normal distribution to draw mutations for the next generation. CMA is a major improvement to the previous algorithm, but it has an unpredictable nature and is still somewhat sensitive to local optima.

Natural Evolution Strategies keep the correlated mutations of CMA, but also try to reduce the sensitivity to local optima. NES estimates a gradient towards better expected fitness in every generation using a Monte Carlo approximation. This gradient is then used to update both the parent individual's parameters and the mutation matrix. NES uses a natural gradient instead of a regular gradient to prevent early convergence to local optima, while also ensuring large update steps. These differences make NES faster and less sensitive to sub optimal solutions compared to CMA.

4.2. Canonical NES

Now that we understand the concept of NES, we can take a look at a basic form of the algorithm.

```
Input: f, $\theta_{\text{init}}$
repeat
    for $k$ = $1..\lambda$ do
        draw sample $z_k \sim \pi(\cdot|\theta)$
        evaluate the fitness $f(z_k)$
        calculate log-derivatives $\nabla_\theta \log\pi(z_k|\theta)$
    end
    $\nabla_\theta J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta) \cdot f(z_k|\theta)$
    $F \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta)\ \nabla_\theta \log\pi(z_k|\theta)^T$
    $\theta \leftarrow \theta + \eta \cdot F^{-1}\nabla_\theta J$
until stopping criterion is met;
```

Algorithm 1: Canonical Natural Evolution Strategies

Algorithm 1 was taken from [15] and depicts a pseudo code for the canonical NES algorithm.

The goal is to compute a gradient over the fitness function with regards to the search distribution θ and use it to update the distribution parameters, which are then used to draw the next generation of samples.

First of all, the inputs are the fitness function (f) and the initial parameters for the distribution (θ). Since NES usually uses the normal distribution, the parameters will be the mean (μ) and the standard deviation (σ).

The first step in the algorithm is to draw all the samples from the normal distribution. In order to do that, we need to evaluate the fitness of each sample and calculate their log-derivatives.

Once we have drawn and evaluated the desired amount of samples (λ) as well as calculated their log-derivatives, we can use that information to calculate the gradient and update the distribution parameters accordingly. These parameters will be used to create the next generation.

Now the only thing that is left is to repeat all the steps until a stopping criterion is met, or in other words, until we have a satisfying solution. This is the basic idea behind NES. In order to use it for task allocation, we will need to make some adjustments.

5. Solving Task Allocation with NES

As described in section 3, solving the Task Allocation Problem, is equivalent to optimizing a function, while staying within specific constraints. This is why using a state of the art function optimization algorithm like NES is a good approach. Unfortunately there is a problem with using the regular NES. It uses a multivariate normal distribution to draw samples, which is a continuous distribution and as such makes the algorithm incompatible with discrete variables. As our formulation from section 3 accepts only discrete solutions, we will need to adjust the algorithm accordingly.

In order to solve this problem, we will use the approach by Benhamou et al. in [16]. They found a way to make CMA compatible with discrete variables. They show in great detail that it is possible to extend the method for drawing samples to multivariate binomial correlated distributions, which are shown to share similar features to the multivariate normal used by CMA and NES. In other words, they show that the multivariate binomial distribution is the discrete counterpart of the multivariate normal distribution.

As described in section 4.1, NES is very similar to CMA. In particular, both use a multivariate normal distribution. The other differences like the natural gradient don't affect this method. So all we have to do is change our algorithm such that mutations are not drawn from a normal distribution, but a multivariate binomial one:

$$\mu + \mathcal{B}(\sigma^2 C) \qquad (5)$$

With μ as the mean, σ as the standard deviation and C as the covariance matrix in the case of CMA-ES, which we will translate to NES.

Now that we have all the puzzle pieces we can put them together and define a basic algorithm. The algorithm itself is very similar to Algorithm 1, but there are some key differences.

```
Input: f, c, $\mu_{\text{init}}$, $\sigma_{\text{init}}$
repeat
    for $k$ = $1..\lambda$ do
        draw sample $z_k \sim \mu + \mathcal{B}(\sigma^2 F)$
        evaluate the fitness
        check constraints c
        calculate log-derivatives
    end
    compute gradients
    update F
    update distribution parameters ($\mu$, $\sigma$)
until stopping criterion is met;
```

Algorithm 2: Task Allocation with NES

Algorithm 2 shows a pseudo code for solving task allocation with NES. As it can be seen, the major differences to Algorithm 1 are first of all the inputs. To solve Task Allocation we need to pass our fitness function (2), but also the constraints (3) and (4). This is necessary as not all valid solutions may satisfy the constraints set by our model. This is also the reason why we need to check the constraints for each sample that we test. Another big difference is the distribution, from which we draw our samples. We are using the binomial distribution for the reasons mentioned before. The other difference is that we left out the exact calculations. They can be taken over from the original algorithm for the most part, but may need some small tweaks, since we are using a binomial distribution and there may be some room for improvement in the algorithm. In the original paper [15] there are already some mentions of the algorithm being tailored to specific use-cases (see sNES in section 3.5). To recognize whether there is room for improvement in our use-case it is necessary to implement and test the algorithm in a common task allocation scenario.

6. Conclusion

We have seen that Task Allocation is a widespread problem. There have been many different approaches to solving it, but due to it being NP-hard, it is very hard to find a suitable one. Most instances of the problem are solved either by the branch-and-bound approach, if the optimal solution is needed, or by heuristics, if smaller deviations from the optimal solution can be tolerated. These heuristics are often tailored to the specific instance of the problem and do not translate well into other instances. In other words, there still does not exist a perfect solution, which solves every instance of the problem optimally. Although we did not test our method, we can take a look at the results in the original NES paper [3]. They have tested NES on many different optimization problems and concluded that, NES can go toe to toe with most other function optimization algorithms. This and the fact that it has a polynomial complexity leads us to believe that it can be as good as, if not better than, currently used algorithms for some instances of the task allocation problem. Naturally, it is necessary to implement the algorithm first, before coming to any further conclusions, but what we can say, is that it is certainly an approach which is worth considering.

7. Future Work

The plan for the future is to first implement the algorithm and test how well it performs. It will almost certainly be necessary to tweak the calculations compared to Algorithm 1, in order to truly optimize it for task allocation. Once the algorithm is implemented and optimized for task allocation, we are confident that it will be a solution which offers both quality and speed. If NES turns out to be as good as we hope, it may be possible to find even more areas where NES could bring an improvement. Task Allocation isn't the only problem, which can be boiled down to a function optimization problem and is usually solved with heuristics. There are innumerable others, which is why the need for quality heuristic approaches is staggeringly big. So the next step after testing NES with Task Allocation is to find other similar problems in need of a better heuristic solution.

References

[1] J. Clausen, "Branch and bound algorithms – principles and examples," 1999.

[2] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '16. New York, NY, USA: ACM, 2016, pp. 69–80. [Online]. Available: http://doi.acm.org/10.1145/2933267.2933312

[3] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 3381–3387.

[4] J. Seeger, A. Bröring, and G. Carle, "Optimally self-healing iot choreographies," 2019.

[5] I. Stanoi, G. Mihaila, C. Lang, and T. Palpanas, "Whitewater: Distributed processing of fast streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, pp. 1214–1226, 10 2007.

[6] S. Rizou, F. Dürr, and K. Rothermel, "Solving the multi-operator placement problem in large-scale operator networks," *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pp. 1–6, 2010.

[7] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004. [Online]. Available: https://doi.org/10.1177/0278364904045564

[8] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 01 1965. [Online]. Available: https://doi.org/10.1093/comjnl/7.4.308

[9] M. Lewis, B. Alidaee, and G. Kochenberger, "Modeling and solving the task allocation problem as an unconstrained quadratic binary program," 04 2004.

[10] J. Cano, D. R. White, A. Bordallo, C. McCreesh, A. L. Michala, J. Singer, and V. Nagarajan, "Solving the task variant allocation problem in distributed robotics," *Autonomous Robots*, vol. 42, no. 7, pp. 1477–1495, Oct 2018. [Online]. Available: https://doi.org/10.1007/s10514-018-9742-5

[11] W.-H. Chen and C.-S. Lin, "A hybrid heuristic to solve a task allocation problem," *Computers & Operations Research*, vol. 27, no. 3, pp. 287 – 303, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054899000453

[12] F. Glover, "Tabu search—part i," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989. [Online]. Available: https://doi.org/10.1287/ijoc.1.3.190

[13] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133 – 137, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016763779390023A

[14] A. K. Gupta and G. W. Greenwood, "Static task allocation using (μ, λ) evolutionary strategies," *Information Sciences*, vol. 94, no. 1, pp. 141 – 150, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020025596000126

[15] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *Journal of Machine Learning Research*, vol. 15, pp. 949–980, 2014. [Online]. Available: http://jmlr.org/papers/v15/wierstra14a.html

[16] E. Benhamou, J. Atif, R. Laraki, and A. Auger, "A discrete version of CMA-ES," *CoRR*, vol. abs/1812.11859, 2018. [Online]. Available: http://arxiv.org/abs/1812.11859

TLS Fingerprinting Techniques

Zlatina Gancheva, Patrick Sattler*, Lars Wüstrich*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga94vad@tum.de, sattler@net.in.tum.de, wuestrich@net.in.tum.de

***Abstract*—Internet security has become a key concern to society in the last couple of decades as more and more sensitive data are being transferred over the Internet. This has led to the adoption of cryptographic protocols such as Secure Sockets Layer protocol (SSL) and Transport Layer Security protocol (TLS), which serve to protect information sent across the Internet. However, even though encryption resolved many security problems, it raised another question and namely how to inspect network traffic while still complying with privacy restrictions.**

TLS Fingerprinting is a method, developed to assist network monitoring. This paper takes a closer look on how TLS Fingerprinting works and analyzes the advantages of it as a client identification method by reviewing different Fingerprinting implementations.

***Index Terms*—Transport Layer Security, Secure Socket Layer, Network monitoring, Client identification, Fingerprinting**

1. Introduction

Nowadays, Transport Layer Security protocol (TLS) is the cryptographic protocol that is used to encrypt the majority of the internet traffic. It creates a huge visibility gap, which serves to prevent third parties from observing user's communication. This, however, poses a challenge to network administrators, who are trying to analyze traffic and who do not necessarily have access to the endpoint devices. The traditional way of intercepting and decrypting traffic is no more applicable, since it does not comply with current privacy standards and causes lower network performance [1]. Therefore, there is a pressing need for a method to improve traffic analysis. A possible solution to this problem is the generation of TLS fingerprints to identify network clients.

TLS fingerprinting - a non-invasive method - meets all the following criteria for traffic monitoring. It aims to provide quick and successful client identification, while being compatible with existing technologies and preserving the integrity of the encrypted information [2], [3]. TLS fingerprinting a is completely passive and payload based approach, which works by capturing and analyzing the unencrypted messages exchanged during a TLS session initialization. In those messages both parties agree on various parameters such as protocol version and encryption keys, which will be used to establish the encrypted connection that follows. This procedure is defined by the term 'handshake' and it is subsequent to the TCP 3-Way Handshake. Most of the handshake parameters are specific enough for a unique client signature to be built and recorded into a database.

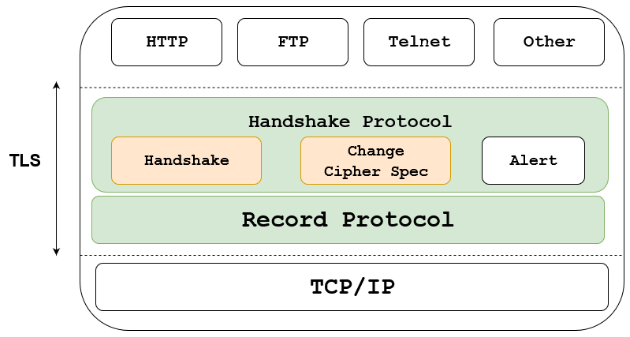


Figure 1: TLS protocol structure [4]–[7]

1.1. Outline

This paper is divided into 5 Sections. The second Section aims to explain TLS fingerprinting by making a detailed observation of the organization of the encryption it is based on. It provides a brief overview of TLS's history, current TLS versions in use and explains in detail important steps such as the TLS client-server Handshake. A detailed overview of different fingerprinting techniques is done in the third Section. The applicability of the results is discussed in Section four. Lastly, Section five concludes the paper.

2. Background

Transport Layer Security (TLS) is a cryptographic protocol, descendant of Secure Socket Layer protocol (SSL), and was first released in January 1999 [1]. Its most widespread version now is TLS 1.2 with more than 95 percent of the web servers supporting it as of February 2020 [8]. In 2018 however, a major TLS upgrade was made and TLSv1.3 was released. It aims to increase speed by reducing the number of handshake messages and improve security by not supporting outdated ciphers and hashing algorithms (e.g. SHA1, MD5, DES) [1]. The 1.3 version is relatively new, but the most popular browsers such as Chrome, Firefox and Opera already support it in their latest releases [9].

TLS should provide [10]–[12]:

- *Authentication* - The server always authenticates itself to the client.

- *Data integrity* - After the connection establishment the data cannot be tampered with by attackers without detection.
- *Confidentiality* - After the connection establishment the data is only visible to the endpoints.

The TLS protocol enables client - server applications to communicate over the network in a manner, designed to prevent interference and eavesdropping. This done by encapsulating and encrypting data from the application layer, which ensures end-to-end security [4]. Hence, TLS is typically implemented on top of TCP with regard to the TCP/IP model (Figure 1). It is the encryption protocol currently standardized [13] for securing the most widespread network protocols, such as HTTP, FTP, SMTP and takes part in VoIP and VPN protocols [14]. As can be seen from Figure 1, the TLS protocol consists of two parts: the Handshake protocol and the Record protocol. The fist one ensures that the communicating parties authenticate themselves and is responsible for the negotiation of cryptographic parameters and key establishment [7]. The second one utilizes the parameters negotiated during the handshake. The Record protocol splits the transferred data into records, which are then individually protected [6]. For the purpose of TLS fingerprinting this paper is going to focus on the Handshake protocol, since as mentioned in the Introduction section only there the information exchanged between the client and the server is in plain format.

However, before going into detail about the handshake procedure, it must be pointed out that TLS uses a combination of both Symmetric and Asymmetric encryption [15]. Asymmetric encryption uses a public-private key pairing, so that data that is encrypted using the public key can only be decrypted using the private key and the other way around [16]. Its purpose is to authenticate the identity of the website's origin server. This is also known as public key encryption. Symmetric encryption on the other hand, uses only one key for encrypting and decrypting data. During the Handshake information is exchanged using asymmetric encryption, until the two sides are finished generating the session keys. Afterwards the session is encrypted using Symmetric encryption.

The Handshake process for TLSv1.3 is graphically presented in Fig 2 and explained as follows [17] :

1) The client calculates a few private/public keypairs for key exchange and requests a TLS Handshake by sending a *ClientHello* message that contains the following cryptographic information:
    - *Preferred TLS version* (TLS 1.3, 1.2, 1.1, etc.)
    - *Client random variable*, which represents a 32 byte string, used to prevent valid data transmission from repetition or delay with malicious intent.
    - *Session ID*, that has the default value of null if this is the first time connecting to this server [10].
    - *Cipher suites list* (e.g. ECDHE, RSA, PSK), ordered by preference of the client. A Cipher suite is a collection of encryption algorithms used to establish a secure connection [**?**].
    - *Compression methods*, used to decrease the bandwidth.
    - *List of Extensions*, which specify additional parameters (e.g. server name, padding). There are about 20 extensions, but among the most prominent ones are Signature Algorithms, Key Share, Elliptic Curves and Elliptic Curve Point Format [18]. They could also be included in the ClientHello fingerprint in order to bring more diversity [19]:
    - *List of public keys*, which contains a list of public keys that the server might find suitable for key exchange
2) The server calculates his private/public key-pair and answers the client with several messages. First is a *ServerHello* message that contains the negotiated protocol version, the chosen cipher suite, the session ID, another random byte string, compression method as well as the public key. The client and the server then both calculate the the shared session key that will be used to encrypt the rest of the handshake, using their private keys and the public key they have received from their partner.
3) The second message from the server is a *ChangeCipherSpec*, which serves to inform the client that from now on the all the messages will be encrypted with the shared key. (however in TLSv1.3 this message is sent simply as a middlebox compatibility mode [20])
4) A *Wrapper* message follows, that comprises of the *Server Encrypted Extensions*, *Server Cerificate*, *Server Certificate Verify* and *Server Finished* messages. The emphasis here falls on the fact that the rest of the handshake communication is encrypted, which is new in TLSv1.3 / is a major upgrade to TLSv1.2.
5) The client also sends a *ChangeCipherSpec*, which has the same purpose as the one send from the servers.
6) Finally the client also sends a *Wrapper* message containing the *Client Finished* message, informing that the handshake was successful for the client.

Now for the duration of the TLS session the server and client can send each other data that are encrypted symmetrically with the shared session key.

3. Fingerprinting

Previously a client used to be identified by the browser User-Agent found in the HTTP header. This is application layer information, which is now encrypted when the client uses cryptographic protocol such as TLS. Nevertheless, careful examination of network traffic has shown that clients can still be identified by capturing the unique set of plain text parameters from the Client- and ServerHello messages. It is important to point out that the elements of the Client- and ServerHello messages stay static throughout different sessions, which allows for previously known clients to be easily recognized. All the client records

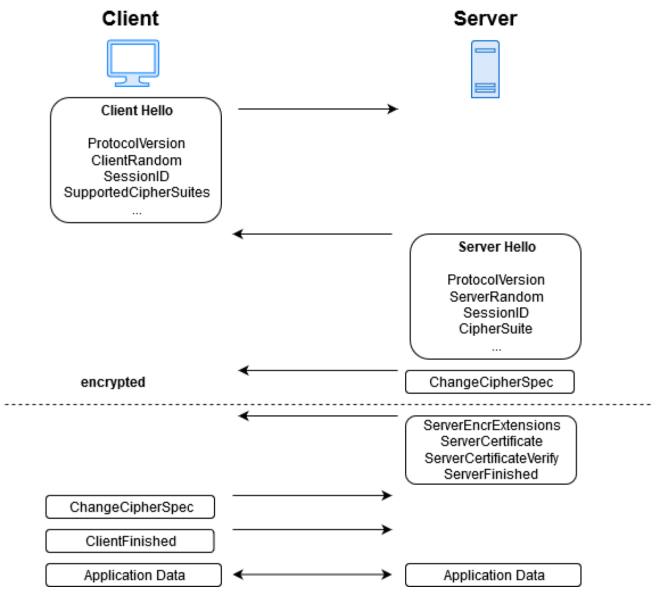


Figure 2: TLS Handhake Steps [1], [17]

are stored in a dictionary database, as this serves to quickly identify known TLS connections and fingerprint new unknown ones. In addition to that, clients with odd behavior can be tracked and discriminated if they are found to be malware applications. Since malware is known to use quite unique/custom parameters when they use TLS communication (normally old or obsolete TLS versions and/or small number of extensions or cipher suites) [2c03], a blacklist with their fingerprints can be composed to aid various TLS fingerprinting implementations.

3.1. Fingerprinting Methodologies

Numerous studies have worked with ClientHello packets to fingerprint TLS. In 2009 Ristić et al. [21] analyzed how to fingerprint SSL/TLS clients by evaluating the Handshake parameters, including the Cipher suites and Extensions list [21] [19]. In this Section three fingerprinting techniques using TLS implementations will be described.

Network-based HTTPS Client identification – this traffic analysis technique achieves proper client identification by creating a dictionary, where the Cipher suite list of the client is paired with their respective User-Agent. The list of Cipher suites is chosen over other elements from the TLS handshake, for it is the most diverse amongst the parameters, supposedly specific enough to identify a client. Other elements of the Handshake have only a few different values and are therefore found not suitable.

This method is based on a combination of two approaches [4]:

- *Host-based* - based on server monitoring - measures connections using the decrypted information from a HTTPS connection, such as the HTTP header, once it is received on the server side. The main advantage of this technique is that it provides results with high accuracy and is applicable in a controlled environment. It is however, also dependable on the amount of clients accessing the monitored server and there is no guarantee for diverse enough traffic. This could result in an insufficient amount of produced pairs.
  Essentially the accuracy of the data depends on the attractiveness of the server [4].
- *Flow-based* - based on network monitoring – this method works on the precondition that clients use both HTTP and HTTPS protocols when they communicate with the server. Thus it scans the traffic for connections that share the same IP source address. Then select a cipher suite list from the HTTPS connection and pairs it with the User-Agent from the HTTP connection, which is the closest in time [4], [14]. As opposed to the host-based approach, the flow-based one is not limited to a single server, so it provides more diverse pairs. Key weakness of the flow-based method is that is could provide ambiguous/perplexing results, because there are usually more than one User-Agent corresponding to a Cipher suite list [4]. Normally the User-Agents should have only slight differences, like software version. So the ones that deviate notably are supposedly connections, forged by web crawlers, pretending to be a legitimate clients. Therefore only the most similar User-Agents sub-strings were taken [14]. There are some possibilities to improve this method [14]. The first one is to manually inspect the pairs, which is still an approach prone to errors and time consuming. The second option is to repeat the measurement. Yet repeating it in a different time window or with different network settings would not necessarily provide complimentary results. Another way to fixing this shortcoming is to extend/spread the fingerprint to the TCP/IP layer.

Combined, the host- and flow- based approaches were proven to be sufficient for the creation of a usable dictionary. Such dictionaries must contain about 300 cipher suit lists with their assigned User-Agents in order to be reliable [14]. After careful examination of the results provided both methods show that the top 10 cipher suite lists covered more that 68 percent of the network traffic and the top 31 cipher suite lists are enough to represent about 90 percent of the traffic. This shows that using both method it is feasible to identify clients with high accuracy.

JA3/JA3S fingerprinting - this technique is a project from Salesforce [1], which utilizes both the ClientHello and the ServerHello to fingerprint the negotiation between client and server, using MD5 hash to produce an 32 character fingerprint, that is easy to digest. [22], [23].

Initially there was JA3, where only the client side of the TLS session establishing messages where exploited. It composes a client fingerprint by collects the decimal values of the bytes for the following fields in the ClientHello packet: Protocol version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats [24]. It then joins those values together in a string, ordered as listed above, using commas to separate the field and a dash to part each value in each field. If there are no TLS Extensions in the ClientHello, the fields are left empty [23]. Those values are captured at the earliest possible stage, even before the server responds. This results in a very large fingerprint, which is why the strings are hashed using MD5 hash.

Alone JA3 is not always enough to create an unique fingerprint, because when client applications use the same OS sockets or common libraries their JA3 fingerprints will be identical. A resolution to this shortcoming is the extension of JA3 - JA3S [1].

JA3S essentially does the same thing JA3 does, but with the server response – it uses the ServerHello packet to gather information from the following fields: Version, Accepted Ciphers, and List of Extensions [22], [23]. Then it concatenates them. This is useful, since servers reply to different clients differently, but to one client the same way in every session. JA3/JA3S provides additional benefits to the detection of malware. For example, if the JA3 fingerprint of the malicious application looks indistinguishable from a JA3 fingerprint of a legitimate application, so it can only be recognized from the server's response. Hence, the combined usage of JA3+JA3S contributes to a highly trustworthy identification / results in a more accurate malware detection [1], [22], [23].

Lastly, JA3/JA3S also has some disadvantages. The first one is that the MD5 hash has become obsolete [1]. It is important to clarify that back in 2017-2018 developers chose this hash type, because then it was supported by current technologies. However, as mentioned in the Background section, this is no longer the case, for MD5 is no longer supported in TLSv1.3, which urges the hash type of the JA3/JA3S to be changed. Additionally the JA3/JA3S technology is blacklist-based, which implies that its trustworthiness depends on how often the blacklist is updated.

Markov Chain Fingerprinting – this traffic classifications technique is conducted on the server side and is designed based on the message type sequence that emerges in a single-direction flow form the server to the client. It can use first-order or second-order homogeneous Markov chains to model statistical TLS fingerprint of different applications [25].

Fundamentally, Markov chains are utilized when computing the probabilities of certain events by viewing them as states transitioning into new or past states [26]. At the beginning of the method development, researchers used first-order homogeneous Markov chain model for computational simplicity [25]. This technique operates under the assumption that the parameters of each TLS session differ considerably and therefore the fingerprint of each application is distinctive enough. However, due to the limited amount of states during a TLS session, it could happen, that many applications contain alike transitions in their fingerprints, which could cause them to be misclassified. This problem could be avoided if the technique is upgraded to second-order Markov chains that are able to capture more diverse application features, which further balances the relationship between complexity and truthfulness [25].

There are a couple limitations to this method. The first one being, that applications change their TLS session initialization parameters over time, which means that for higher accuracy levels, it is advised that application fingerprints must to be updated periodically [27]. The second one being, that the technique struggles to recognize applications that have not taken part in the training stage. To resolve this issue, new application fingerprints must be incorporated in the existing database accordingly regularly [25].

Overall, the Markov chain fingerprinting technique results in proper applications discrimination, which can come from one of the following reasons [27]:

- incorrect and diverse implementation practices
- the misuse of TLS protocol
- various server configurations
- the application nature.

This leads us to the conclusion that proper classification could possibly be avoided by omitting implementation mistakes or creating the secure layer on a limited set [27].

4. Discussion

TLS Fingerprinting amongst other ways of client fingerprinting is a reliable method for traffic analysis and client identification [28]. It is passive, payload based and requires no endpoint agent data [29]. Nevertheless there could occur some inconveniences, such as collisions.

Fingerprint collision is the event of two fingerprints, belonging to different applications, overlapping [29]. The solution to collision avoidance is to take as many parameters from the ClientHello message as possible. Suitable candidates for that are extensions, such as Signature algorithms, Elliptic curves and Elliptic curve point format [29]. This offers greater variety in comparison to assessing cipher suites alone.

A different kind of inconvenience is the fact that TLS Fingerprinting implementations can be avoided or redirected. Based on the researches of Husak et al. [14] and Frolov et al. [19], a client can prevent TLS Fingerprinting in the following ways: by using a proxy, by manual change in the Cipher suite list or by mimicking popular TLS implementations.

- Usage of proxy redirects the TLS fingerprinting technology to fingerprint the cipher suite of the proxy instead of the one of the client [14]. However, the cipher suite list of a proxy could already exist in the fingerprint database and thus be recognized and associated accordingly.
- Manual changes in the client Cipher suite list are usually done by forced reducing of the list [14]. The client continues to communicate with a reduced Cipher suite list and is therefore not recognized as an existing record. So the Fingerprinting technique fails to find the corresponding User-Agent. But reducing it so much as to forge another client's cipher suite list is a quite difficult.
- Mimicking TLS implementations such as browsers. Mimicking also has its challenges - it is difficult to keep up with the rapidly-changing TLS browser implementations and their many features. It is also difficult to know what types of fingerprints to mimic.

Generally fingerprint collisions and TLS fingerprinting avoidance techniques are not a obstacle for the majority of fingerprinting tools. The biggest concern of TLS fingerprinting remains the database that the tools use, because TLS fingerprinting is only as good as the database supporting it. Dictionaries may turn into disadvantage, if they are hard to maintain and update. Currently the process of the creation of data set collection was manual, but there are ongoing researches, attempting to automatize it in the future. [30].

Classification tools usually require to be trained on a particular data set consisting of benign traffic, which must be updated regularly to ensure novelty data. Especially hard to maintain and update is the collection of malware samples [2c03].

5. Conclusion

Encryption of data is crucial when aiming to protect the privacy of users. In modern networks, the TLS protocol is the current encryption standard for data transferred over the Internet. Although it is used to mask the plain text information from the application layer, TLS also provides a set of unique observable parameters that allow many conclusions to be made about both the client and the server [1].

In this paper we have reviewed the three most widely spread/diverse techniques used for TLS fingerprinting, starting with the simplest one – Network-based HTTPS Client identification – essentially divided into two approaches, which are both based on the extraction of the the most varied components from the TLS session initialization messages and writing them down in a database. The second one being the JA3/JA3S that is partially based on the Network-based identification as it upgrades it through memory optimization, hashing the values into 32-character unique fingerprint, making it quicker for malware software to be recognized. The last and most complicated method is the creation of a fingerprint using homogeneous Markov chains (either first or second order) so as to simulate the time-varying message sequence that occurs during the TLS session initialization. Vital characteristic trait of this method is that conducted on the server side and focuses mainly on detecting abnormal TLS sessions and improving discrimination practices. All of these techniques can identify clients with high accuracy while sustaining their privacy. A comparison based on the statistical accuracy of these techniques is hard to derive, because experiments with each one of them has been done individually, over different amounts of time, using different traffic samples.

In the future it would be interesting to conduct an experiment to test how these three techniques would perform under the same set of conditions (e.g. time window, network and servers).

Overall, TLS fingerprinting is a subsection of passive client identification and traffic. There are other methods for client fingerprinting, that may partially incorporate the TLS technology (for example OS fingerprinting [31], web browser fingerprinting, website fingerprinting, signal fingerprinting, cookies [32]) that are efficient as well.

References

[1] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)." [Online]. Available: http://arxiv.org/abs/1607.01639

[2] L. Brotherston, "synackpse/tls-fingerprinting," accessed: 2020-01-23. [Online]. Available: https://github.com/synackpse/tls-fingerprinting

[3] The generation and use of TLS fingerprints. Accessed: 2020-01-23. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=539893

[4] M. Husak, M. Cermak, T. Jirsik, and P. Celeda, "Network-based HTTPS client identification using SSL/TLS fingerprinting," in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, pp. 389–396. [Online]. Available: http://ieeexplore.ieee.org/document/7299941/

[5] Transport layer security protocol | microsoft docs. Accessed: 2020-01-18. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786441(v\%3Dws.11)

[6] M. D. Center. TLS record protocol - win32 apps. Accessed: 2020-01-23. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/secauthn/tls-record-protocol

[7] ——. TLS handshake protocol - win32 apps. Accessed: 2020-01-23. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/secauthn/tls-handshake-protocol

[8] Qualys SSL labs - SSL pulse. Accessed: 2020-02-23. [Online]. Available: https://www.ssllabs.com/ssl-pulse/

[9] Can i use... support tables for HTML5, CSS3, etc. Accessed: 2020-02-23. [Online]. Available: https://caniuse.com/#feat=tls1-3

[10] RFC 8446 - the transport layer security (TLS) protocol version 1.3. Accessed: 2019-12-13. [Online]. Available: https://tools.ietf.org/html/rfc8446#section-4.1.2

[11] RFC 5246 - the transport layer security (TLS) protocol version 1.2. Accessed: 2019-12-13. [Online]. Available: https://tools.ietf.org/html/rfc5246

[12] L. Brotherston, "Lee brotherston's work," accessed: 2019-12-13. [Online]. Available: https://github.com/synackpse/tls-fingerprinting

[13] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of age: A longitudinal study of TLS deployment," in *Proceedings of the Internet Measurement Conference 2018 on - IMC '18*. ACM Press, pp. 415–428, accessed: 2019-11-18. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3278532.3278568

[14] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," vol. 2016, no. 1, p. 6. [Online]. Available: https://doi.org/10.1186/s13635-016-0030-7

[15] An overview of the SSL or TLS handshake. Accessed: 2019-12-14. [Online]. Available: www.ibm.com/support/knowledgecenter/en/ssfksj_7.1.0/com.ibm.mq.doc/sy10660_.htm

[16] Comparative study of symmetric and asymmetric cryptography techniques | semantic scholar. Accessed: 2019-12-13. [Online]. Available: https://www.semanticscholar.org/paper/Comparative-Study-of-Symmetric-and-Asymmetric-Tripathi-Agrawal/e0e4810c5276f9c05cc82425fcf911f206c52bef

[17] The illustrated TLS 1.3 connection: Every byte explained. Accessed: 2020-01-18. [Online]. Available: https://tls13.ulfheim.net/

[18] TLSfingerprint.io - extensions. Accessed: 2019-12-13. [Online]. Available: https://tlsfingerprint.io/top/extensions

[19] S. Frolov and E. Wustrow, "The use of TLS in censorship circumvention," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_03B-2-1_Frolov_paper.pdf

[20] Middlebox compatibility mode. Accessed: 2020-01-18. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rzain/rzainmiddlebox.htm

[21] Ivan ristić: HTTP client fingerprinting using SSL handshake analysis. Accessed: 2019-12-11. [Online]. Available: https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html

[22] Open sourcing JA3 - salesforce engineering. Accessed: 2019-12-13. [Online]. Available: https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41

[23] TLS fingerprinting with JA3 and JA3s - salesforce engineering. Accessed: 2019-12-13. [Online]. Available: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[24] B. Vasudevan, "Elliptic curves in transport layer security (TLS) - a presentation tutorial," p. 4.

[25] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bigrams," vol. 12, no. 8, pp. 1830–1843.

[26] K. Chan, C. Lenard, and T. Mills, "An introduction to markov chains."

[27] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, pp. 781–789. [Online]. Available: http://ieeexplore.ieee.org/document/6848005/

[28] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "Web tracking: Mechanisms, implications, and defenses." [Online]. Available: http://arxiv.org/abs/1507.07872

[29] SquareLemon. Accessed: 2019-11-17. [Online]. Available: https://blog.squarelemon.com/tls-fingerprinting/

[30] TLS fingerprinting in the real world. Accessed: 2019-12-13. [Online]. Available: https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world

[31] [1706.08003] OS fingerprinting: New techniques and a study of information gain and obfuscation. Accessed: 2019-12-13. [Online]. Available: https://arxiv.org/abs/1706.08003

[32] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, ISSN: 2157-4960.

Building an OS Image for Deep Learning

Daniel Gunzinger, Benedikt Jaeger*, Sebastian Gallenmüller*
**Chair of Network Architectures and Services, Department of Informatics*
Technical University of Munich, Germany
Email: daniel.gunzinger@tum.de, jaeger@net.in.tum.de, gallenmu@net.in.tum.de

***Abstract*—This paper documents the process of creating an reproducible OS image for deep learning. The target framework is Tensorflow 2, which is provided in a recent version in this image, accompanied by support for GPU acceleration, which can improve performance and thus development time significantly. During the creation of this OS image, a number of problems have been encountered and solved. Due to the created OS image, it is now possible to rapidly spin up a server, which can immediately provide a reproducible environment fit for development and training of deep learning applications that utilize the Tensorflow 2 framework with GPU acceleration.**

***Index Terms*—operating system, tensorflow, gpu, deep learning, cuda, cudnn**

1. Introduction

In this paper the process of building an OS Image based on Debian Bullseye in order to provide GPU support for deep learning applications using Tensorflow 2 with the new generation of Nvidia RTX Super graphics cards is discussed.

This aims to provide a reliable and reproducible OS image for the development and training of deep learning applications, reducing development time by removing tedious setup tasks and improving execution performance of the developed applications with GPU acceleration.

The OS image also aids the reliability of testing results of any executed applications as the exact configuration of the whole OS image can be preserved and tests can be repeated by loading the OS image again on the same target network node. Thereby providing the same hardware and software environment and thus a reproducible platform that can help in achieving reliable application testing results for all users of the OS image.

This paper is structured as follows, in Section 2 background information about the used software and hardware is provided, including the infrastructure setup which the OS image is targeted for. In Section 3 the building process of the OS image is detailed, this contians information about the testing and building methodology and the problems that were encountered during this process. Section 4 provides an overview of the performance testing that was done including benchmark results for the used test scripts on a target server and its hardware specifications.

1.1. Deep Learning

Deep Learning is a type of machine learning which can assign a given input to a set of known elements based on previous training.

Deep Learning is often used to classify images or recognize known elements in a given input image, but it is not constrained to only work on images, but can also work on other input data formats such as text or sound as DeepL[1] or speech recognition software show.

However in this work, the focus is to provide GPU acceleration of deep learning frameworks primarily concerned with image recognition.

For this purpose it is required to create a model with a number of different layers which process the input image and pass their output on into the next layer.

There are different layer types which perform different computations on the input they recieve and also differ in their output and output formats.

One layer type which is commonly used in image recognition models are convolutional layers, this layer type performs operations on a given range of the input matrix. In this type of layer each output is the result of a function over a window of the input.

Another commonly used layer type are fully connected layers, where each input is connected to each output by a weight. In this type of layer each output is the result of an operation involving every single input and its weight.

One more important layer type are pooling layers which are used to filter the input, perform a specific operation on a range of inputs and combine them into one output.

2. Background

In this section the targeted infrastructure setup is detailed, and choices for software and hardware are explained.

2.1. Infrastructure Setup

The targeted infrastructure setup consists of a server network which is managed through the pos software, which is an abbreviation for plain orchestrating service.

Among many other features out of scope regarding this paper, it allows for the use of a calendar. In this calendar multiple users working with the server network can enter dates, durations and targeted servers in order to reserve them for their later use, and allows for an overview of the current reservation status of the available servers. When a user of this system has a ongoing reservation of one or multiple servers, it can be used to allocate the specified

1. https://www.deepl.com/press.html

servers. Once one or multiple servers are allocated by a user, they can be started, stopped and reset individually. Available OS images can be chosen for deployment and an overview of their operational status can be displayed.

For the image building process the mandelstamm software is used to create OS images that can be used in the pos software mentioned above. It consists of a set of build scripts which are used to specify the contents of the OS image, such as the base operating system and software that is supposed to be installed once the image is executed, and to configure the image for deployment.

The pos and mandelstamm software are projects by the Chair of Network Architectures and Services and are not publicly available.

2.2. Software

In this section the software components of the OS image are described.

As we need a reliable operating system, Debian was chosen since it provides a stable base system with a broad range of software packages being available for installation and use through the repositories. In particular Debian Bullseye (11), the current testing testing release has been chosen.

As the main focus of this work is to provide an OS image for GPU accelerated deep learning, deep learning libraries need to be available too. Thus Tensorflow 2 and PyTorch are installed through the pip package manager and included in the OS image.

While Tensorflow 2 provides APIs in multiple different programming languages[2], we focused on providing support for a Python workflow, which is also required for PyTorch.

Python is available in many different versions, with incompatibilities between the major versions 2 and 3, for our OS image we aimed to provide a recent version of Python 3.7, which is available through the Debian repositories and is also supported for use with the Tensorflow 2 library.

As a major task of this work is to provide GPU acceleration for Tensorflow and PyTorch, the GPU driver and multiple libraries from Nvidia also need to be included into the OS image.

The installed version of the drivers is 430.64, with CUDA 10.1 being provided by libcuda1 and cuDNN 7.6.5.32 installed through Nvidias generic package for linux targets. Other important libraries in order to provide GPU acceleration are libcudart10.1, libcublas, libcufft, libcurand, libcusolver, libcusparse and the nvidia-cuda-toolkit.

2.3. Hardware

The decision to work with the Nvidia graphic cards stems from their hardware acceleration capabilites for compute and deep learning applications.

The amount of streaming processors present on these cards is useable through the CUDA API which can provide impressive speedups for Tensorflow programs over execution on general purpose CPUs. Another feature which the Nvidia RTX series GPUs provide are the Tensor Cores, which can provide another speedup over common general purpose hardware for mixed precision matrix multiplications, commonly used in the network training phase.

2. https://www.tensorflow.org/api_docs

3. Building Process

In order to build the images the mandelstamm software is used to create and package the OS images.

For the first attempt at creating the target OS image Debian Buster was chosen as the operating system, as it is the latest stable release version. In order to enable support for the used GPUs the nvidia-driver and the nvidia-smi (system management interface) were included into the OS image.

When attempting to test the created image on the targeted server, it became apparant that the version of the nvidia-driver package available in Debian Buster is not recent enough to support the used RTX 2080 Super GPUs, as elaborated upon in Section 3.2.1.

Thus the built image was tested on a server containing two Nvidia GTX 1080 Ti GPUs in order to determine if the approach for driver installation had succeeded. The installation success could be confirmed by executing the nvidia-smi utility which reported the two GPUs with their correct names and further statistics such as current power consumption and memory usage.

The next step was to install Tensorflow 2, which instead of building it from source, can be acquired via the Python pip utility. During the installation of Tensorflow 2 the latest version available through pip was used, at the time of this testing this was version 2.0.0, which was released on September 30th of 2019[3].

This installation led to the discovery of the next problem due to Tensorflow 2 requiring at least version 19 of the pip utility, which is not provided in any of the Debian repositories as described in Section 3.2.2.

This required the pip utility to not be installed through the apt package manager using the Debian repositories, but instead through downloading a provided installation file from the Python Package Authority (PyPA) and executing it in order to complete the installation[4].

Thereafter the installation of Tensorflow 2 was again attempted by installing version 2.0.0 through the pip utility, this time completing successfully and thus enabling first tests to be run.

Since the goal of this image is to provide GPU acceleration support with the Tensorflow 2 library, the first test was to see if the GPU is recognized as a processing device by Tensorflow 2 as described in Section 3.1.1.

This revealed warnings about a number of additional libraries needed in order to register the GPU successfully, and libcudnn.

All of these libraries except for libcudnn are available through the Debian repositories, however installing them was of no help since the available versions did not match the required CUDA version, as described in Section 3.2.3.

At this point the upcoming Bullseye release of Debian, version 11, has been chosen due to the package availability problems and a lack of driver support for the targeted

3. https://pypi.org/project/tensorflow/#history
4. https://pip.pypa.io/en/stable/installing/

graphics card series that were encountered with Debian Buster.

As the Bullseye release of Debian is still in development, neither the official release date nor the end of life date is known yet, however extrapolating from the Debian version history[5] a release date in 2021 and an end of life date around 2024 would match the current pace of releases.

By changing the target OS to Debian Bullseye a problem with the `mandelstamm` build scripts became apparent, as the URL format for the security repositories had changed for Debian Bullseye as elaborated upon in Section 3.2.4. Thus the `mandelstamm` build scripts had to be adapted in order to successfully build the OS image.

After this problem was addressed, the build script for Debian Bullseye was modified by adding calls to the `apt` package manager in order to install the Nvidia GPU driver and system management interface from the Debian repositories. Another call to the package manager was added in order to install the aforementioned additional GPU libraries which are fortunately available in the required and matching version in the Debian Bullseye repositories.

Afterwards the built Debian Bullseye image was deployed to the test server with the Nvidia RTX 2080 Super GPU, and the `nvidia-smi` command was called in order to determine correct installation of the GPU drivers. This could be confirmed as the `nvidia-smi` utility did now provide the correct name of the card in the output alongside the other statistics, it also reported both the `nvidia-smi` and driver version as 430.64, which officially supports the targeted card as listed in the release notes for this driver[6].

After all necessary tools and libraries have been successfully installed Tensorflow 2 can be installed via `pip`, however the version that is going to be installed has to match the installed CUDA and cuDNN version. In our case the chosen version was version `2.1.0rc0`, which is the first release candidate of Tensorflow version 2.1 and requires CUDA 10.1 and cuDNN >=7.4[7].

After the first confirmation of driver support for the installed card the next testing stage was executed by listing available devices in Tensorflow as described in more detail in Section 3.1.1. This returned the expected result of a processing device list with one CPU and one GPU device being available.

By running the test script in order to verify the availability of GPU acceleration a problem with Tensorflows memory allocation behaviour on GPUs became apparent, which is described in greater detail in Section 3.2.6.

After solving the GPU memory allocation issue, all parts of the test script could be executed successfully on both CPU and GPU, demonstrating full functionality of the OS image.

3.1. Testing and Deployment

This section explains the methods used for testing the functionality of the produced OS images.

5. https://wiki.debian.org/DebianReleases
6. https://www.nvidia.com/Download/driverResults.aspx/153714/en-us
7. https://www.tensorflow.org/install/source#gpu

3.1.1. Initial testing by listing devices. For initial testing the following two lines of code were used to list the available processing devices, such as the CPU and GPU.

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

3.1.2. Deep learning test script. In order to test the functionality of the Tensorflow installation and to ensure that the installed CUDA and cuDNN versions work with the chosen version of Tensorflow a custom test script has been created.

It trains a neural network for image classification using the CIFAR10 dataset[8], and is structurally similar to a Tensorflow tutorial example for CNNs[9].

The script has two major sections, in the first section convolutional layers are used, this section can be disabled and thus skipped. This first section contains two Conv2D layers[10] with a Max-Pooling layer in between the two Conv2D layers.

The second section of the script uses fully connected layers, thus the input is first flattened. After the input has been flattened two fully connected layers are added as Dense layers[11], with the first layer using a ReLU activation function and the second (final) layer using the softmax activation function.

Using these layers as described above, the model is then trained for ten epochs over all images contained in the CIFAR10 dataset.

3.2. Encountered Problems

This section elaborates on the problems that were encountered during the creation and testing of the OS images and their solutions.

3.2.1. Target GPU not supported in Debian Buster. Due to the first build targeting Debian Buster, the latest version of the driver available in the Buster-specific repositories was installed, which was version 418.74.

However as we need to support an Nvidia RTX 2080 Super GPU this is not recent enough, as it does not support any of the Super-series cards, which were released in July of 2019[12].

This was noticed due to the nvidia-smi utility output not reporting the name of the installed GPU correctly.

3.2.2. Tensorflow software requirements. There are also problems with the availability of recent Python3 `pip` versions on several Debian versions including Buster and Bullseye, as the repositories only provide `pip` version 18.1, yet at least version 19 is required for our target application Tensorflow 2.

8. https://www.tensorflow.org/datasets/catalog/cifar10
9. https://www.tensorflow.org/tutorials/images/cnn
10. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
11. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
12. https://www.anandtech.com/show/14663/the-nvidia-geforce-rtx-2080-super-review

3.2.3. Mismatched GPU library versions in Debian Buster. While the repositories for Debian Buster do contain libraries for CUDA 10.1 support, none of the other important libraries for GPU acceleration support are available in version 10, instead only in version 9.2, which could not be used successfully in combination with Tensorflow 2. The problematic libraries are `libcudart`, `libcublas`, `libcufft`, `libcurand`, `libcusolver` and `libcusparse`.

3.2.4. Missing support for Debian Bullseye in mandelstamm. As `mandelstamm` does not have a specific build script for Debian Bullseye, an image creation was first attempted by copying the build script for Debian Buster and changing the release name from Buster to Bullseye.

This however did not result in a successful image creation as the security repositories could not be found. After closer inspection of the build scripts and Debian documentation, an adjustment had to be made to the generic Debian build script as the URL of the security repository had its format changed[13], thereby creating a special case in the build script.

3.2.5. Installation of the cuDNN Package. An important library regarding GPU acceleration for deep neural networks is Nvidias cuDNN package. It is only available through a repository for Ubuntu, installations on other distributions require a more generic package available for download on Nvidias website through the Nvidia Developer Program[14], which however requires a membership in the mentioned developer program.

Thus it is necessary to install the package manually according to Nvidias documentation[15].

3.2.6. Issues with cuDNN and Tensorflows default settings. When executing the test script on a GPU device, an error about not being able to allocate memory was returned. This turned out to be a configuration issue instead of a driver or library issue and has been solved by adding a small loop which iterates over the available GPU devices and calls the following function for each GPU device:

```
tf.config.experimental.set_memory_growth(device,
True)
```

After setting this flag for each GPU device the convolutional network part of the test script could be run without issues on the GPU device, which as described in Chapter 4 allowed for a significant speedup over executing it on the CPU.

3.2.7. Default build options of available Tensorflow packages. During the execution of the test scripts on the CPU using the Tensorflow `2.1.0rc0` build obtained via the `pip` package manager the following warning was logged:

```
Your CPU supports instructions that this
TensorFlow binary was not compiled to use:
AVX2 AVX512F FMA
```

Which implies that the execution times observed using when the CPU for the test script could be significantly lowered by building Tensorflow from source with enabled support for the advanced AVX instruction sets and the fused multiply-add instuction set, which both can accelerate a common operations of deep learning applications significantly.

This is a problematic default build setting as a majority of recent CPU architectures, starting with Intels Haswell microarchitecture in June 2013[16] include support for the AVX2 and FMA instruction sets.

If support for using these instruction sets is added in future builds a meaningful speedup could be observed when running deep learning applications without GPU acceleration, as the SIMD instruction sets AVX2 and AVX512 can improve the throughput of matrix multiplications and other common operations in multiple neural network types.

4. Performance Evaluation

In this section we compare the performance of the test script when running on the CPU and the GPU.

The performance testing was conducted on a server with the relevant specifications listed in Table 1.

TABLE 1: Testing server specifications

Part	Equipped
Processor	Intel Xeon Silver 4214 (12c24t, up to 3.20GHz)
Memory	8x32GiB DDR4 2400MHz in hexa-channel
Mainboard	Supermicro X11SPi-TF
Graphics card	Nvidia RTX 2080 Super

The test script (see Section 3.1.2) was used and the complete execution time for each configuration was obtained with the time command.

In order to show the speedup of different operations, two different configurations were used for the test script execution. First, the complete script including all operations was executed. For the second configuration the test script was modified to skip the execution of convolutional operations.

Both of these configurations were executed three times with and without GPU acceleration and the execution time for these tests was then averaged in order to alleviate the effects of run-to-run variance.

TABLE 2: Performance testing results

Test	CPU execution time	GPU execution time
Complete	763.65 (14945.69)	90.73 (156.78)
Fully connected network	107.00 (1439.07)	63.49 (126.29)

Results format: real time (user time) in seconds

The results listed in Table 2 show that GPU acceleration provides a significant speedup, especially when working with convolutional networks, which can be explained by inspecting the functionality of the layers and the capabilities of the used hardware.

13. https://www.debian.org/releases/bullseye/errata#security
14. https://developer.nvidia.com/rdp/cudnn-download
15. https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html

16. https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)#New_instructions

Some of the operations in the script described in Section 3.1.2 are rather compute bound, while other types of operations, such as pooling or the processing of activation functions is rather memory bandwidth bound[17]. The parallelizability of the layers is an important aspect of the observed performance scaling, as the amount of compute cores differs greatly between the used CPU, where 12 cores with 24 threads are available, and the used GPU, where 3072 shader processors and 384 tensor cores are ready to compute.

In memory bound operations the GPU will also have an advantage as it features a 256bit wide memory bus operating at 15500MT/s resulting in a theoretical peak bandwidth of 496GB/s to the GDDR6 chips. In comparison the CPU can access six channels of DDR4 memory with a width of 64bits each, resulting in a 384bit wide memory bus operating at 2400MT/s resulting in a theoretical peak bandwidth of 115.2GB/s.

With a complete execution of the test script we can observe a speedup of 88.1% when enabling the GPU acceleration over a CPU only execution. By disabling the execution of the convolutional layers, the difference in execution time shrinks significantly, however enabling GPU acceleration still yields a 40.7% decrease in runtime.

With these numbers we can also see that the execution of convolutional networks profits much more from GPU acceleration, as the execution time compared to the reduced test configuration increases by 42.9% while the time taken when executing on the CPU increases by 613.7%.

However, it is important to note that the chosen release for Tensorflow 2 (`2.1.0rc0`), seems to lack support for CPU instruction sets that could improve the execution time when running on the CPU significantly, as described in Section 3.2.7.

5. Conclusion and Future Work

The created OS image supports the use of GPU acceleration with Tensorflow 2, which provides a significant reduction in runtime for deep learning applications, especially in applications which include convolutional neural networks.

Currently the OS image is based on Debian Bullseye for the operating system, featuring the Nvidia drivers in version 430.64 for support of their latest series of graphics cards, as well as a number of accompanying libraries.

Other installed software includes Python 3.7, a recent version of the `pip` package manager and most importantly recent versions of Tensorflow 2 and PyTorch.

With the OS image ready for deployment, time is saved in the development workflow as tedious setup tasks can be skipped by deploying the OS image to an available server and using it for the development tasks.

The building process did also show that in order to create an environment featuring recent versions of the Tensorflow and PyTorch frameworks with GPU acceleration support, special attention needs to be brought to the used graphics cards and the GPU driver version, as well as the available libraries regarding GPU acceleration, as these libraries have dependencies on specific CUDA versions.

For the future the OS image can be extended to include more available deep learning frameworks as well as more software tools that ease the development workflow.

Additional images using other operating systems, e.g. Ubuntu as a basis could also be created in order to expand the available software support through more repositories, thus allowing for more use cases and letting developers choose an environment with which they are already familiar.

If a version of this image is to be created for CPU only execution of Tensorflow 2 applications, it would be beneficial to check the target CPU for its supported instruction sets, compile Tensorflow from the source code. By including all instruction sets that the target CPU can support in the compilation settings a performance advantage over the precompiled binaries available through `pip` can be achieved.

17. https://docs.nvidia.com/deeplearning/sdk/dl-performance-guide/index.html

Modern Traceroute Variants and Adaptations

Julius Niedworok, Johannes Zirngibl, Patrick Sattler*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: julius.niedworok@tum.de, zirngibl@net.in.tum.de, sattler@net.in.tum.de

Abstract—**Traceroute is a widely used tool to perform measurements on the network path from a source to a destination. However, there are some issues when applying it to the recent structure of the Internet. Load balancing techniques are very common to encounter on today's Internet paths. This article describes a few approaches that can be used to adapt the original concept of traceroute to the current structure of the Internet. When looking at network topology discovery, the *Multipath Detection Algorithm (MDA)* and its lightweight version MDA-Lite come in place. For broader discovery *Yelling at Random Routers Progressively (Yarrp)* can help to speed up the process, which is advantageous for measurement of short living network topologies. However, when tracing a single application flow, Service traceroute provides more precise results. Choosing the right approach for a given use case is crucial in order to obtain appropriate results.**

Index Terms—**Active Internet Measurements, Traceroute, Network Topology Analysis**

1. Introduction

Over the years traceroute became a well-known tool for network administrators to perform network diagnosis tasks. The original approach introduced by V. Jacobson in 1989 [1] is implemented in the standard Linux tool *traceroute*. The idea of traceroute is to provide the user insights on the path, which is taken through a network to a given destination. It creates active measurement probes for each hop along the path. This is done iteratively by increasing the probe's *Time To Live (TTL)* and inspect the ICMP response message of the host where the TTL expired [2]. The probe packets can be of different protocol types depending on the use case and on the network infrastructure. Although traceroute is widely used today, it does not fit the needs of the present structure of the Internet. Traditional traceroute is based on the assumption that there is a single forward path to the destination host. As shown in Figure 1, this can be problematic in load balanced networks. Traceroute would send a probe to node *A* with TTL n. When increasing the TTL to $n+1$, the probe could either traverse the upper or the lower path. In case of Figure 1 the probe reaches node *B*. Traceroute again increases the TTL to $n+2$. However, this time the probe traverses the lower path, through node *C*, and discovers node *F*. Given that the previous probe discovered node *B*, traceroute would assume that there exists a path between node *B* and *F*. The ICMP response message contains only

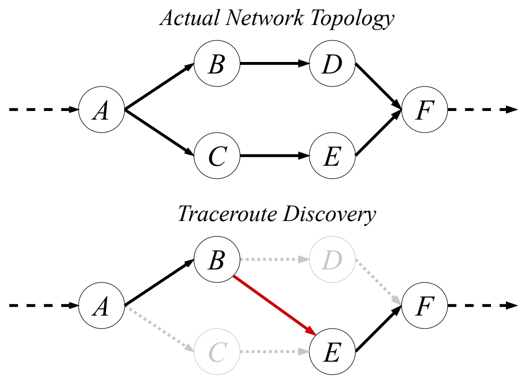


Figure 1: Problems using Traditional traceroute

the address of the node where the TTL expired. Therefore, traditional traceroute cannot detect, if the probe traversed through a different path than previous probes. Moreover, nodes *C* and *D* remain hidden.

The utilization of load balancing techniques has increased excessively over the last years [3]. In general there are different ways of performing load balancing on network traffic [4], [5]:

- **Per-destination** load balancing depends on the destination specified in the packet.
- **Per-flow** load balancing derives its decision from the packets flow identifier.
- **Per-packet** load balancing concentrates on keeping the load as equal as possible. No effort is made to keep packets of a single flow on the same path.

Per-destination load balancing does not create a problem when tracing the forward path to a single destination host. Traditional traceroute changes the header to be able to match an ICMP response packet to its corresponding probe packet [2]. This is done by varying the destination port field for UDP probes and the sequence number for ICMP Echo probes. In case of per-flow load balancing such behavior leads to potential different paths for each probe packet. As described by Augustin et al. [6], this can be mitigated using Paris traceroute. Paris traceroute explicitly controls the packet header to direct the packet through a certain path. An ICMP response packet contains the discarded header of the probe, as well as the first eight octets of the payload. Instead of changin the flow identifier, Paris traceroute makes use of these eight octets for matching probes to the responses. In consequence, Paris traceroute can deal much better with topologies such as the one shown in Figure 1. Paris traceroute cannot get around the problems created by per-packet load balancing due to its randomness. However, in those cases where there is per-packet load balancing employed Paris traceroute can detect it.

These days traceroute is not only used for what it was initially intended. It is applied to a broader range of problems, i.e., detection of all load balanced paths or tracing specific services. Some approaches even separate from the idea of tracing towards a single destination host and rather try to get a bigger picture of the network topology [7], [8]. The following sections will look at recent traceroute-based approaches that deal with these additional use cases.

2. Network Discovery

This section describes approaches to discover the network in a broader sense. First, it explains an approach to discover all paths from a single source to a single destination. Afterwards, it diverges from the traditional scenario of tracing towards a single destination. A recent approach to discover the network topology is presented.

2.1. Detecting all Paths

After introducing Paris traceroute, the authors explored a way of getting a broader view of the network topology [9]. In 2009 Darryl Veitch et al. [7] present their final version of the *Multipath Detection Algorithm (MDA)*. MDA extends Paris traceroute to reveal all possible load balanced paths to a given destination. It runs iteratively through the paths and elicits all interfaces at each hop. In order to enumerate the paths from a node at hop h, it generates a number of probes with random flow identifiers and selects the ones which reach that node. Subsequently, it sends these probes to hop $h+1$ to discover all successors. It sends these probes under the assumption that hop h is a load balancer that evenly allocates traffic to k paths. MDA is using a statistical approach to compute the number of probes n_k which need to be sent over the node at hop h in order to enumerate all of its successors at a given level of confidence. If it is not possible to find more than k successors after sending n_k probes, MDA stops and assumes to have enumerated all possible successors. However, in case $k+1$ successors have been found, MDA continues with the assumption that there are at most $k+1$ successors. Correspondingly, it will generate and send n_{k+1} probes. With this approach MDA claims to find all load balanced paths between a source and a destination host.

There are different constructs that can be encountered when discovering the network topology. As soon as load balancing comes in place, so-called, *diamonds* will be exposed. According to the definition of Augustin et al. "a diamond is a subgraph delimited by a *divergence point* followed, two or more hops later, by a *convergence point*, with the requirement that all flows from source to destination flow through both points" [10]. Figure 2 shows two examples of diamonds. The upper diamond is an example of an *unmeshed* diamond and the lower one an example of a *meshed* diamond. Vermeulen et al. [3] define *meshing* between two hops to meet one of the following criteria:

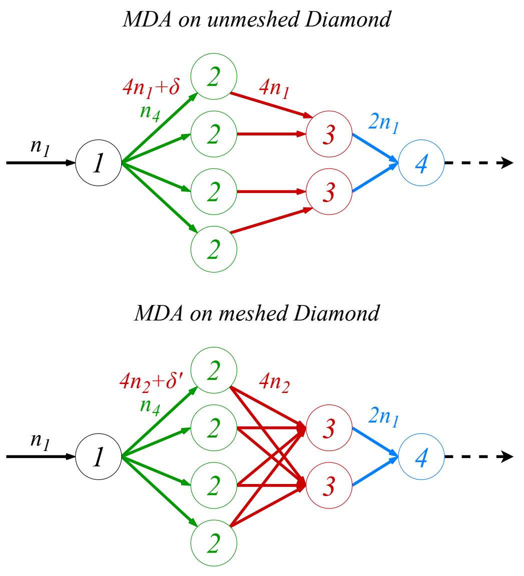


Figure 2: MDA on Meshed and Unmeshed Diamonds

- If the two hops h and $h+1$ have the same number of nodes, then the out-degree of at least one node of hop h must be two or more. Consequently, at least one node at hop $h+1$ will have an in-degree of two or more.
- If hop h has fewer nodes than hop $h+1$, then the in-degree of at least one node at hop $h+1$ must be two or more.
- If hop h has more nodes than hop $h+1$, then the out-degree of at least one node at hop h must be two or more.

A diamond is considered to be meshed as soon as one hop is meshed. In the example of Figure 2 MDA would execute the following steps in order to detect the diamond topology [7]:

1) To discover hop 1, MDA sends n_1 probes with the assumption that there is only one node. As it cannot discover a second node, it will stop after n_1 probes.
2) In order to discover hop 2, MDA will send n_1 probes again. However, this time it will discover another node before it stops. Therefore, it will continue with the assumption that there are two nodes. While sending n_2, probes it will discover a third node and, after adjusting the number of probes, a fourth one. As it cannot discover a fifth node, it will stop after sending n_4 probes.
3) When discovering hop 3, MDA starts with the assumption that each of the four nodes has one successor. A number of n_1 probes need to be sent over each of the four nodes. As the set of n_4 probes that revealed these four nodes at hop 2 does not contain a number of n_1 probes per hop 2 node, MDA needs to generate more probe packets. When generating new probes, it is unlikely that the new probes are evenly spread over all four nodes. Therefore, an overhead of a few additional probes, denoted as δ will occur. In the case of the unmeshed diamond, MDA will stop after sending these probes as only one

successor node is discovered for each node at hop 2. For a meshed diamond, however, MDA will detect a second node. It needs to send n_2 probes over each node at hop 2. Therefore, MDA generates another set of additional probes with a potential overhead of δ'. As no third node will be discovered, MDA will stop after sending these probes.

4) In the last step MDA will discover the node at hop 4 by sending a number of n_1 probes per node at hop 3. As there are already n_1 probes available per hop 3 node, there is no need to generate new probes. MDA will stop after sending a total number of $2n_1$ probes.

This procedure requires a lot of active measurement traffic to be generated. For this reason, Vermeulen et al. [3] came up with a lightweight version of the algorithm, called *MDA-Lite*. It makes use of the benefit that nearly half of all diamonds, which are found in the Internet, only have a single hop with multiple nodes [3]. In consequence, meshed diamonds are very uncommon. This allows MDA-Lite to minimize the situations where it needs to generate new probes for already discovered nodes, as seen in Step 3 of full MDA. Figure 3 illustrates the process of MDA-Lite on the same unmeshed diamond as in Figure 2. Discovery of hop 1 and 2 works the same way as for MDA. When revealing hop 3, the benefit of MDA-Lite comes into effect. Instead of generating and sending $4n_1 + \delta$ probes, MDA-Lite will consider the set of nodes at hop 2 as one node and proceed as usual. As two nodes are discovered, n_2 probes need to be sent. These probes can be taken from the set of n_4 probes, which were sent before. Similar to the third step, MDA-Lite will consider all nodes at hop 3 as one and send a total number of n_1 probes to discover hop 4. MDA-Lite will therefore send $2n_1 + n_2 + n_4$ probes in total. In contrast, full MDA will send $11n_1 + \delta$ probes for the unmeshed and $8n_2 + 3n_1 + \delta'$ probes for the meshed diamond. Keeping in mind that for most confidence levels, including the ones used by Vermeulen et al., $n_2 < 2n_1$ holds, this clearly shows the amount of probes saved by MDA-Lite. In their paper Vermeulen et al. compare the savings in more detail based on actual numbers by defining example values for all parameters of MDA-Lite [3].

The above steps of MDA-Lite do not reveal all edges of the graph even in unmeshed diamonds. However, the task to obtain the rest of the edges is deterministic rather than stochastic. It will therefore most likely require less probes for high levels of confidence. The three possible situations are handled as follows [3]:

- In case there are more nodes at hop h than at hop $h + 1$, additional probes are generated for each node at hop h and sent to hop $h + 1$. This will find all successors for the nodes at hop h and thus find the missing edges.
- If there are more nodes at hop $h+1$ than at hop h, the probes that have discovered the nodes at hop $h + 1$ will be sent to hop h. This will unmask the missing edges.
- Given that both hops have the same amount of edges, both of the above procedures are applied.

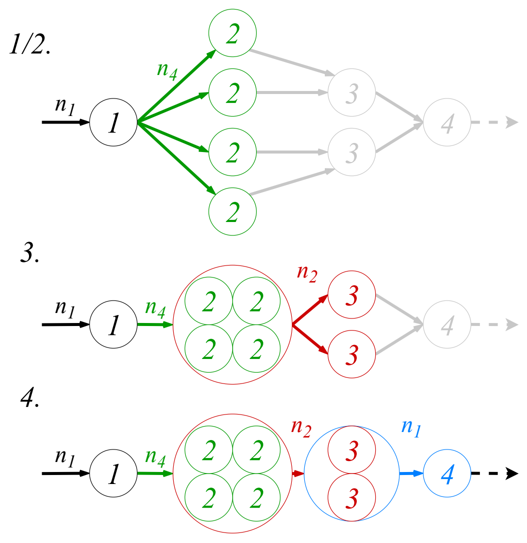


Figure 3: MDA-Lite on Unmeshed Diamond

Assuming there is no meshing, MDA-Lite will perform with much less probes than full MDA. In order to detect meshing MDA-Lite, applies stochastic probing for potential meshing. This includes the need for the costly generation of additional probes. The amount of these additional probes can be set by the user as a parameter introduced by MDA-Lite. For all meaningful values used by Vermeulen et al. the amount of additional generated probes is less than with full MDA [3]. In case meshing is detected, MDA-Lite will switch to full MDA.

2.2. Large Network Topology Discovery

Augustin et al. [10] already came up with the idea of getting a broader view on the network topology. They started using MDA to trace traffic to a network prefix rather than an address. However, the problem of extensive active measurement network traffic gets even more significant when tracing towards a network prefix. When requiring a lot of time to complete the measurement of a network topology, one can encounter changes of the topology, which will bias the result of the measurement. Due to that problem, Beverly introduced an implementation called *Yelling at Random Routers Progressively (Yarrp)* [8]. He determined that the problems of most traceroute implementations, when it comes to larger network topology discovery, to reside in the following:

- **Maintenance of State** for each probe that has not yet received a response: this state usually consists of timing information as well as the probes identifier.
- **Sequential Probing** of the path: this can lead to a significant execution time for large networks. Moreover, sending probes sequentially, hop by hop and node by node, can lead to intrusion detection mechanisms to identify the probe traffic as a security risk.

Yarrp deals with the sequential probing problem by adjusting the order of the probes by randomizing the destination and TTLs of the probes. To achieve pseudo randomness, the target IP addresses, as well as the TTLs, are encrypted by a keyed block cipher. This projects the input of IP addresses and TTLs onto a new pseudo random order. To eliminate the need to maintain state for each probe, Yarrp encodes all state information into the probe packet. This is done by overloading the TCP header of the probe. For example, the TTL of the original probe gets encoded in the IP identification field [11] and the elapsed time resides in the TCP sequence number. The approach of Yarrp can be beneficial, especially in situations when the measurement time is important [8]. It allows measurement and analysis of short living topologies.

3. Service Oriented Approach

Tracing the path of a specific application's network traffic can be hard using (Paris) traceroute. The application can have multiple flows to different destination hosts. In order to solve this task with Paris traceroute, one would have to create probe packets with flow identifiers that match the flow identifiers of the applications network traffic [12]. But even if the flow identifier matches the applications flow identifier, for TCP connections many routers discard packets that do not belong to an already known open connection [13]. The idea that many service oriented tracing tools apply is that they place their probe packets into an existing flow. There are different implementations available for TCP connections [13]–[15]. However, they all work under the assumption that there is a single connection that is established for each application. A lot of applications fetch their content from multiple sources over different connections. This makes it hard to observe the paths, which the network traffic of an application is taking through the Internet.

For this reason, Morandi et al. [16] proposed an implementation called *Service traceroute*. This tool uses the idea of previous service oriented implementations, but it provides a feature to automatically select all flows to trace based on high level user input and provides the capability to trace them simultaneously. For example, the user can specify to trace all flows related to traffic of the streaming service *Youtube* by using its name as a service specifier. Moreover, the tool also works with UDP.

The execution of Service traceroute is split into two phases:

- The **observation phase** identifies the flows to be traced by observing the specified network interface. The flows are identified by the high level search terms of the user. To achieve this kind of functionality, a database of services signatures was created and is used to identify the services.
- The **path tracing phase** executes the probing on each identified flow. It stops probing as soon as the application closes the TCP connection or after a given timeout for UDP.

The probe packets are constructed from the observed packets of the actual application flow. For TCP packets, Service traceroute uses empty TCP Acknowledgements with the same flow identifier and sequence number as probe packets. In order to match the ICMP response messages to the probes, it uses the IP Identification field [11]. For UDP packets, Service traceroute constructs UDP packets with an empty payload using the same flow identifier as in the original packets.

Service traceroute can be a useful tool, especially when tracing short living flows, such as small web downloads [16]. On the one hand, the probe traffic will result in a higher overhead in these situations. On the other hand, other Tracing tools could be hard to launch in these situations, as they do not provide any reasonable fast way of identifying which flows to trace. As part of their work, Morandi et al. compared Service traceroute to Paris traceroute with and without MDA [16]. For Paris traceroute they used the same flow identifier as the flow identifier of the application traffic. The results show that 7% of the discovered paths by Service traceroute could not be discovered by Paris traceroute. When using MDA to discover all paths, more than 40% of the paths reported by Service traceroute are not found within all paths reported by MDA. The authors explain the reason for Paris traceroute performing worse than Service traceroute with the possibility that some routers drop packets that do not belong to an active flow. This proves the need for tools that inject their probes into active flows.

Morandi et al. also inspected potential interference with the application of the observed flows. They did not find any conflict with the packets being sent by the application.

4. Conclusion and Outlook

As the use cases of traceroute slowly expanded to a broad range of problems, the choice of the right implementation is important. When finding all possible existing paths towards a single destination, no matter what the packet's payload will look like, then MDA and its lightweight version MDA-Lite will be the right choice [3]. As a tool for large network topology discovery Yarrp tries to speed up the execution time. This makes it particularly useful to discover short living network topologies [8]. In case of tracing down a single application, the user might not necessarily be interested in parts of the network that are not taken by the applications network traffic. In this case the use of service oriented approaches provides the best results [16].

Besides the approaches discussed in this article, there exist many more solutions. Most traceroute-based solutions do not investigate the use of the *Record Route* field of the IP header [17]. This field is actually meant to allow tracing the paths of network traffic. Each router along the path will include its address in a list inside the IP header. *Tcpsidecar* is a tool integrating the Record Route field [14]. Unfortunately, the amount of hops being captured by the Record Route field is limited and most routers do not adhere to its specification. Moreover, some firewalls even block packets that have the record route option specified.

Instead of actually improving the core behavior of traceroute, there are tools available which try to focus on filtering out the wrong paths reported by traceroute [18], [19].

There are also a few attempts to deploy active measurement infrastructure to deal with the problem of measuring the Internet. One of the most famous ones is *RIPE Atlas* [20]. Relying on global infrastructure brings some benefits as well as some other challenges, which are not discussed as part of this paper.

All in all, traceroute remains a useful tool for ad-hoc measurements. There are a few issues with the traditional approach that can be mitigated by choosing the right improvement.

References

[1] *MAN(8) Traceroute for Linux*, Linux 5.4, 2019, accessed Dec 09 2019. [Online]. Available: http://man7.org/linux/man-pages/man8/traceroute.8.html

[2] D. Malone and M. Luckie, "Analysis of ICMP Quotations," in *Passive and Active Network Measurement*, S. Uhlig, K. Papagiannaki, and O. Bonaventure, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 228–232.

[3] K. Vermeulen, S. D. Strowes, O. Fourmaux, and T. Friedman, "Multilevel MDA-Lite Paris Traceroute," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 29–42.

[4] Juniper, "Configuring Per-Packet Load Balancing," 2018, accessed Dec 09 2019. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/policy-configuring-per-packet-load-balancing.html

[5] Cisco, "How Does Load Balancing Work?" 2015, accessed Dec 09 2019. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html

[6] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 153–158.

[7] D. Veitch, B. Augustin, R. Teixeira, and T. Friedman, "Failure Control in Multipath Route Tracing," in *IEEE INFOCOM 2009*, April 2009, pp. 1395–1403.

[8] R. Beverly, "Yarrp'Ing the Internet: Randomized High-Speed Active Topology Discovery," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: ACM, 2016, pp. 413–420.

[9] B. Augustin, T. Friedman, and R. Teixeira, "Multipath tracing with Paris traceroute," in *2007 Workshop on End-to-End Monitoring Techniques and Services*, May 2007, pp. 1–8.

[10] B. Augustin, T. Friedman, and R. Teixeira, "Measuring Load-balanced Paths in the Internet," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 149–160.

[11] J. Touch, "Updated Specification of the IPv4 ID Field," Internet Requests for Comments, RFC 6864, February 2013, accessed Dec 09 2019. [Online]. Available: https://tools.ietf.org/html/rfc6864

[12] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute Probe Method and Forward IP Path Inference," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '08. New York, NY, USA: ACM, 2008, pp. 311–324.

[13] J. Edge, "Tracing behind the firewall," *LWN.net*, Jan 2007, accessed Dec 09 2019. [Online]. Available: https://lwn.net/Articles/217076/

[14] R. Sherwood and N. Spring, "Touring the Internet in a TCP Sidecar," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 339–344.

[15] D. Kaminsky, *Paratrace Man Page*, Linux 5.4, 2019, accessed Dec 09 2019. [Online]. Available: https://man.cx/paratrace

[16] I. Morandi, F. Bronzino, F. Bronzino, and S. Sundaresan, "Service Traceroute: Tracing Paths of Application Flows," in *Passive and Active Measurement*, D. Choffnes and M. Barcellos, Eds. Cham: Springer International Publishing, 2019, pp. 116–128.

[17] J. Postel, "Internet Protocol," Internet Requests for Comments, RFC 791, September 1981, accessed Dec 09 2019. [Online]. Available: https://tools.ietf.org/html/rfc791

[18] A. Marder and J. M. Smith, "MAP-IT: Multipass Accurate Passive Inferences from Traceroute," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: ACM, 2016, pp. 397–411.

[19] N. Brownlee, "On Searching for Patterns in Traceroute Responses," in *Passive and Active Measurement*, M. Faloutsos and A. Kuzmanovic, Eds. Cham: Springer International Publishing, 2014, pp. 67–76.

[20] "Ripe Atlas," RIPE Network Coordination Centre, accessed Dec 09 2019. [Online]. Available: https://atlas.ripe.net/

Smart-M3 vs. VSL for IoT

Ilena Pesheva, Christian Lübben*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: pesheva@in.tum.de, luebben@net.in.tum.de

***Abstract*—The benefits of autonomous data exchange in software environments and multi-device communication have triggered the development of various middleware services to enhance data exchange in distributed systems. These middleware solutions have gained great importance in reducing overall system complexity and enabling interoperability in the context of information sharing.**

This paper takes a closer look at two data exchange middleware solutions: The Smart-M3 platform and VSL overlay for Internet of Things. They both solve key challenges in the environment of device heterogeneity and propose a data-centric approach to information exchange.

We explain the core differences in their key features and give an overview of their field of application.

***Index Terms*—middleware, data-centric, interoperability**

1. Introduction

In the era of rapid digitalization and constant emerging of new technological devices, the idea of seamless and wireless information exchange between these devices has evolved to a need. This is referred to as ubiquitous computing, which is gradually emerging as the dominant type of computer access [1]. As M. Weiser states in [1], it is enabling nothing fundamentally new, but by making everything faster and easier to do, it is transforming the perception of what is possible. An example is the Internet of Things (IoT) that quickly gained importance as part of the Internet mainly because of its immense impact, remarkable growth and capability. Its main features are interrelating "computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction" [2]. Benefiting from ICN-principles IoT has a data-centric traffic design rather than addressing a specific host. This results in improvements in data latency, scalability, reliability, resilience and is more energy efficient than typical host-based communication [3].

The distributed nature of the devices participating in an IoT system and the heterogeneity of application scenarios introduce interoperability problems and challenges for developers. This triggered the introduction of the Virtual State Layer (VSL). It is, as M.-O. Pahl, S. Liebald, and C. Lübben specify in [3], "a data-centric middleware that securely unifies the access to distributed heterogeneous IoT components.". Its core tasks are to discover, read and write data that another IoT component has produced and thus to be able to orchestrate IoT environments.

The idea behind it is Service Oriented Architecture (SOA). It modularizes the complex IoT applications into smaller mashups, creating microservices that are simple and can be later reused. It fully separates logic and data in IoT services which contributes to major VSL features such as delivering service data even when a service is offline or security-by-design, meaning that security is fully implemented within the VSL [4].

The interest of getting computing devices to interoperate and to do so with whatever devices are locally close at any point in time has also raised the question of dealing with complexity and interoperability issues. Participating in different domains means implementing several different standards, which are often serving specific use cases and not aiming to create a general interoperability framework [5]. The semantic web is attempting to defy this issue by implementing an interoperability framework with the ultimate goal of making a machine understand the World Wide Web data. The result would be a "giant global graph" of linked data that describes the information of the web in a standard model for data interchange - Resource Description Framework (RDF) [6] and Web Ontology Language (OWL) [7] that enable the encoding of semantics [8]. Nonetheless, there is a need for a mechanism that enables sharing of dynamic, rapidly changing information on a local level about the current state of a device and the web is not the most suitable environment for that purpose. Therefore, the interoperability platform Smart-M3 was created with the goal, as explained in [5], "to (enable devices) to easily share and access local semantic information, while also allowing access to the locally relevant parts of the 'giant global graph' to be available". Smart-M3 acts as a smart space solution that enables devices of all kinds to engage in interoperability and have a shared view of data and services. The goal is to provide a better user experience where users can effortlessly add or remove devices from the platform and grant every participating device the same information access.

The similarities between the VSL and Smart-M3 platforms in their main purpose and idea raises the importance to differentiate them and thus to be able to apply them accordingly.

2. Feature comparison

Both VSL and Smart-M3 platforms serve as distributed systems middleware and share the common purpose of enabling interoperability in heterogeneous environments. However, they rely on different architectural approaches and comprehend data in a different manner. The following section gives a brief overview of the general system architecture of Smart-M3 and VSL. Furthermore, we address their core differences considering the following key feature components: data access, data storage, data discovery, data transport, semantic structure and security. This step-by-step comparison will allow us to obtain a full perspective of the functional and semantic divergence of each system.

2.1. General system architecture

Smart-M3 works on the basis of a blackboard architectural model and implements the idea of space-based computing. The architecture it implements consists of two core components: knowledge processors (KP) and semantic information brokers (SIB) that may be concrete or virtual. The core of the system is hosted by a device which contains the SIB and the physical data base. Then there are other devices hosting KPs - pieces of software implemented to read and contribute data to a SIB [9]. One or more SIBs connected to each other define a smart space that contains information provided by the KPs. An illustration example of a M3 smart space distribution is presented in Figure 1 featured in [10] by J. Honkola, H. Laine, R. Brown and O. Tyrkkö.

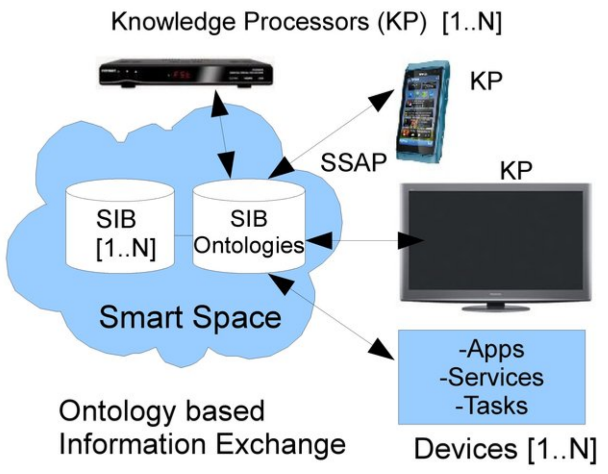


Figure 1: Smart-M3: SIB and KP distribution

The VSL overlays key components are the so-called Knowledge Agents (KA) that manage data for different services. Every KA serves the purpose of storing relevant data for a specific node while also enabling inter-service communication [3]. Figure 2 described in [3] gives a detailed view of the VSL architecture model, consisting of a hardware underlay with IoT nodes, VSL Peer-to-Peer overlay and multiple microservices that register at a KA. The VSL offers IoT nodes unified access to the Knowledge Agents, which can also run on the same IoT node like a service [4].

2.2. Data access

The connection between the SIBs in Smart-M3 is enabled by a protocol providing distributed deductive closure

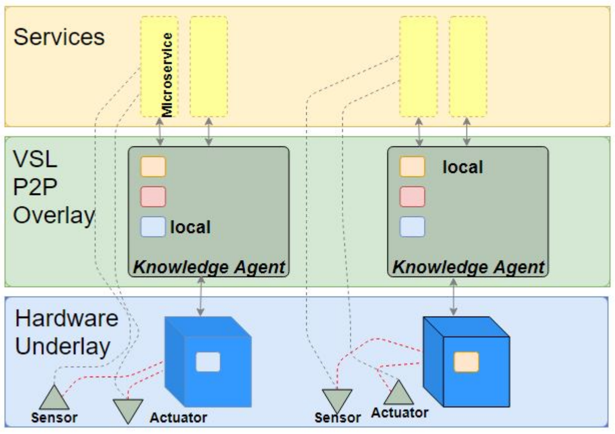


Figure 2: VSL general architecture and logical connectivity

[11]. This allows all KPs to access the same information in the smart space with no regard of the specific SIB they are connected to.

[5] A KP can access a SIB by using Smart Space Access Protocol (SSAP). This protocol has eight operations: *join, leave, insert, remove, update, query, subscribe* and *unsubscribe*. Therefore, the KPs are able to interact with the content in the smart space. The operations enabling this interaction are not concrete as they depend on the defined parameters and the actions that SIB and KP should initiate. They may also be encoded using different formats, JSON or XML for example.

[5] The protocol uses sessions to establish a connection between the KP and the SIB components. First, a *join* operation is executed by the KP, the SIB then inspects it and decides whether the KP can join or not. Following a successful join the KP is allowed to perform other activities. If the SSAP protocol is successfully supported by the SIB and KP implementations interoperability will be ensured.

The VSL, as mentioned in the Introduction, finds its specific purpose in Internet of Things services. Its working principle is to fully decouple data from hosts providing ICN properties [12]. Unlike Smart-M3, it is implemented as self-organizing Peer-to-Peer overlay, enabling data-centric communication between and within IoT hosts. It targets microservice-based architecture in order to run independent IoT microservices. Access is based on hierarchical data item identifiers and is enabled via get and set, subscriptions to changes and stream connections [4]. This is to be differentiated from the Smart-M3 data access method, which is push-based one instead of a publish-subscribe one as in VSL.

2.3. Data Storage

Smart-M3, as already stated in Section 2.1, has two key components - the knowledge processor and the semantic information broker. The information is stored in the smart space, consisting of one or more SIBs, as an RDF graph. The storage is realized on some defined ontology, however, there is no obligation for using a specific one. The KPs, having once successfully accessed the smart space, are then able to contribute to or to read the stored in the smart space content. [5]

VSL works on a different principle for storing data (see Figure 2). Instead of having a distributed shared memory architecture that contains all the information like the smart space in the Smart-M3 platform (see Figure 1), it stores data always at the source. This follows as a consequence of the disconnection between logic and data [4]. The platform implements data managing agents, the KAs, which are connected to each other in a Peer-to-Peer structure. Each of them is responsible for running a number of microservices and stores data relevant to them. This results in distributing information to be stored and retrieved all over the network. Therefore, locality is an important issue when dealing with VSL, especially when information exchange happens constantly. However, due to the full location transparency in the data lookup process, data can be stored on any KA, not necessarily on the one that is running on the same IoT node (source) as the service.

2.4. Data Discovery

The VSL, as already mentioned in Section 2.1, implements data discovery in a Peer-to-Peer manner between the KA peers. The KAs are assigned with overlay IDs and an underlying address that can be IP-Address. The data node discovery happens via special tags, provided by the KAs, which return all instance addresses associated with the given tag. This means that the semantic lookup happens KA-locally via a search for coupling candidates. Services do not bind statically, unlike how the Knowledge Processors in Smart-M3 bind to their services.

In contrast to the encapsulated nature of the VSL data discovery that is fully integrated and closed to the concrete IoT network, the Smart-M3 platform is allowed access to parts of the "giant global graph" that results from the Semantic Web [8], in addition to sharing and accessing local semantic information between the engaged software entities and devices. Thus, it is making use of both local and global information, represented as an RDF graph. This allows easy linking of data between different ontologies, which aims to solve the interoperability issue. Unlike the Semantic Web, which represents the idea of a single, centralized web of machine-understandable information, Smart-M3 sets distinct spots, the Knowledge Processors, in the Web. These spots may be connected to many devices of different kinds and gather specific machine-understandable information that is unique but non-exclusive for the particular KP and has a concrete focus and purpose. Overlapping of information between the KPs is even needed to ensure interoperability [5].

2.5. Data Transport

[5] In the Smart-M3 interoperability platform, the core component responsible for data interaction is the SIB. Its internal architecture consists of five layers: Transport layer, Operation handling layer, Graph operations layer, Triple operations layer and Persistent storage layer. The transport layer, being the first access point of the SIB architecture, must ensure that various domains, service architectures and networks are able to communicate and exchange information, regardless of their different communication capabilities. To be able to overcome this issue, the SIB supports various communication mechanisms, such as Bluetooth, TCP/IP, HTTP and NoTA. The most suitable mechanism is being selected depending on the operating environment. This indicates once more one of the core principles in Smart-M3 - to be able to operate regardless of the communication mechanisms restrictions.

[4] As already mentioned in Section 2.3, VSL organizes its information exchange in a Peer-to-Peer process by distributing it between a number of Knowledge Agents. Thus they must be capable of addressing each other accordingly. The information exchange is enabled via IP unicast and multicast connections. Unicast is used in the case of a single recipient of the data and multicast - as the core maintenance of the overlay. The Knowledge Agent has a Transport Manager, which along with the Connection Manager and the Overlay Manager, manages the connectivity between the IoT nodes. It uses HTTP over TCP/IP as a transport protocol, although different protocols are also applicable.

In contrast to Smart-M3, VSL does not adopt any other data transport technology, such as Bluetooth for example, as the communication transport happens within the VSL overlay. It does not have the need to accommodate to the different capabilities of other domains or service architectures like Smart-M3 does.

2.6. Semantic Structure

[5] The Smart-M3 platform allows storing and querying information on the basis of tuple space mechanisms. This means that data is exchanged between a consumer and a producer entity. In the case of Smart-M3 these are the SIB and the KP respectively. Data is produced in tuple form and retrieved from the consumer using a specific pattern. As mentioned in Section 2.5, there are five logical layers responsible for the access, operations and storage of data to the SIB. After access has been established, requests in the form of SSAP operations run in threads to query, insert or remove information from the RDF store [6]. This is handled by the Graph operations layer where the operations are being scheduled. In the Triple operations layer happens the inserting, querying and removing of triplets from the RDF store. Triplets, connected to form a graph, represent the architecture or the RDF data format. The linking of triplets results in a structured data graph, resembling the World Wide Web. A triple is a statement in the form subject-predicate-object (e.g the sky /subject/ has the color /predicate/ blue /object/). The RDF syntax is abstract, meaning that it can be represented by using a variety of notations in the arrangement order subject-predicate-object. The semantic is the representation of the subject, predicate and object roles in the statement. Due to the RDF format of data its linking under different ontologies has been made extremely easy, reducing system complexity and enabling cross-domain interoperability.

[3] Similarly to the Smart-M3 platform, VSL also implements a tuple-space mechanism in the form of a structured data item graph to organize its data, although in a hierarchical manner. Each time a service is being registered at a Knowledge Agent, an identifier is passed for the data model representation. An instance of this

model is being created at the KA allowing the connection and communication of this service to other services via the KA API. The data nodes participating in the structured graph are in fact the digital data twins of the managed IoT software and hardware entities. VSL uses tags and identifiers pinned to the data nodes, offering a modularized tagging approach. The items in the hierarchical structure can be accessed transparently from an arbitrary participating KA.

To summarize, Smart-M3 is using data in the form of triplets according to the RDF syntax to store and retrieve information, whereas the VSL uses a hierarchical data structure in the form of a data node tree and is instantiating digital twins of data every time a service and a KA interact.

2.7. Security

[4] The main issue of the IoT data is security because of the vulnerable nature of private user data. Therefore, VSL implements security-by-design, which means that the mechanisms implemented in the VSL middleware cannot be outmaneuvered. This, as mentioned in Section 2.1, comes as a result of the full separation of service logic and data and promises a secure throughout communication. In particular, VSL assigns certificates to each of its components (services and KAs). These certificates ensure the authentication of software modules, as well as enabling communication between KAs that is TLS-secured, including secure exchange of keys for encrypted stores. Due to access control to IoT nodes and specific synchronization of type information and access modifiers between the KAs, VSL ensures secure addressing and trusted IoT orchestration.

The smart space environment is vulnerable to threats and security risks as well. In contrast to VSL, Smart-M3 has not been provided a sufficient security mechanism to this point. In [13], Kirill Yudenok and Ilya Nikolaevskiy introduce a security solution protecting the data interchange between the KPs and the smart space. For robust authentication they propose the usage of the Host Identity Protocol (HIP) for key exchange [14]. The HIP protocol can be integrated in the SIB access module and thus enable a SIB to restrict access to information in the smart space that the KPs have provided.

2.8. Comparison summary

In order to retain a clear and structured view of the information presented in the previous subsections, Table 1 summarizes and highlights the most important aspects and differences of the Smart-M3 and VSL middlewares.

3. Issues and Reliability

Both the Smart-M3 platform and the VSL overlay aim to implement simplistic architecture designs for reliability and robustness reasons. However, issues and challenges are an inevitable part of every system regardless of its kind. In the following we highlight some important aspects to consider when dealing with data-centric issues and vulnerabilities of each system.

TABLE 1: Key feature differences

	Smart-M3	**VSL**
Data access	SSAP	get/set
Data storage	smart space	at the source
Data discovery	use of local or global information	tag or address based
Data transport	Bluetooth, TCP/IP HTTP and NoTA	HTTP over TCP/IP
Data structure	RDF graph	hierarchical graph
Security	not implemented internally	security-by-design
Implementation	C, C++, Python, C#, Java	Java

3.1. Smart-M3

Arguably the main issue in the Smart-M3 system is security, as mentioned in Section 2.7. There we elaborated the main causes of this issue and mentioned a resolution method presented by K. Yudenok and I. Nikolaevskiy in their work [13]. In [15] written by Matti Eteläperä et al. a test of two smart space information broker implementations is presented: Smart-M3 and RIBS (RDF Information Based System), the second being an M3 tool for devices with restricted computational capabilities. Based on their measurement analysis the authors conclude that neither system is satisfactory enough for wide-spread usage. The authors state that "Smart-M3 performance and usability leave a lot to be desired, as even a simple single triple insert operation has a latency of 86-176 milliseconds in our tests. The performance of Smart-M3 is not suitable for use cases needing fast response times."

3.2. VSL

The Virtual State Layer decouples data not only from hosts (ICN principle), but even from services on hosts, as already mentioned in the Introduction. This presents a significant advantage in the case of a service failure, as decoupled data is then managed only within the middleware. Energy efficiency also follows as a consequence, because not needed services can be interrupted while their data still remains attainable. Nevertheless, the advantages that follow from the ICN principle have challenges on their own. In [16] the authors A. Lindgren, F. B. Abdesslem, et al. address the aspects of naming, caching, actuation, decoupling between publisher and consumer, etc. They suggest that naming can become a size-problem when e.g. the size of the name can become larger that the size of the data; Caching reduces latency but can also be useless when using At Most Once object requesting strategy; Actuation may conflict with the ICN addressing design and further reduce caching advantages and impose latency requirements; Although having multiple advantages, decoupling publisher/consumer has trouble in resolving publisher mobility when deducing the name of the data for consumers.

4. Field of application

In [5] the authors present several smart space application scenarios. They involve different applications, various services and multiple types of devices (e.g. phones, laptops, sensors). All domains have M3 software agents installed on them. An example described in [17] shows a scenario that involves an application for sports tracking, a music streaming service, a gaming domain and a phone call observer application. The results show improved user experience due to seamless component cooperation between the participating devices and services. An ongoing call can trigger information exchange in the smart space resulting in pausing the music and the game. Furthermore, one example demonstrates a mash-up between two different scenarios, proving the ease of their mixing. [5]

An example for the VSL overlay in use is shown in [3] by Marc-Oliver Pahl, Stefan Liebald and Christian Lübben. In their work they present a VSL demo consisting of a smartphone based controller and a light sensor based game whereby they "demonstrate the data-based coupling and the service-orientation of the VSL" [3]. Each participating service implements a VSL interface, several microservices and local data items (figure 2). Users make interactive data queries via the smartphone controller and thus switch lights found by type or address.
This demo illustrates the benefits of VSL in environments where decoupling specific services is needed, whereas the Smart-M3 platform, while also demonstrating inter-service communication, points out the ease of creating scenario mashups.

5. Conclusion

This paper reviews the similarities and differences between the Smart-M3 platform and the VSL IoT overlay. Both systems aim to solve the interoperability problem and thus present data-centric solutions for reducing system complexity in heterogeneous environments. However, they employ contrasting approaches and architectural styles, which makes their distinction of great importance. Smart-M3 is a space-based system that enables interoperability through the use of the Semantic Web. It targets multi-device implementation and enables sharing of local information between hosts. VSL follows ICN principles and has its main focus on Internet-of-Things environments where it manages the entire inter-service communication between IoT devices. It implements a Peer-to-Peer architecture for routing and unlike Smart-M3, VSL is making use of the network connectivity between devices to enable communication and data exchange. It implements security-by-design, whereas Smart-M3 has not implemented a specific security mechanism. Thus we have shown that Smart-M3 can be used easily in an environment of different domains; it handles usage and mixing of various scenario instances. The VSL overlay has its specific focus on the IoT environment where it manages each aspect of the communication services.

References

[1] M. Weiser, “The computer for the 21st century,” vol. 3, no. 3, pp. 3–11. [Online]. Available: https://doi.org/10.1145/329124.329126

[2] What is internet of things (IoT)? - definition from WhatIs.com. (Date accessed: 30.11.2019). [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[3] M.-O. Pahl, S. Liebald, and C. Lübben, “VSL: A data-centric internet of things overlay,” in *2019 International Conference on Networked Systems (NetSys)*, pp. 1–3.

[4] M.-O. Pahl and S. Liebald, “Information-centric IoT middleware overlay: VSL,” in *2019 International Conference on Networked Systems (NetSys)*. IEEE, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8854515/

[5] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, “Smart-m3 information sharing platform,” in *The IEEE symposium on Computers and Communications*, pp. 1041–1046, ISSN: 1530-1346.

[6] RDF - semantic web standards. (Date accessed: 12.01.2019). [Online]. Available: https://www.w3.org/RDF/

[7] “OWL web ontology language overview,” p. 22, (Date accessed: 19.02.2020). [Online]. Available: https://www.w3.org/TR/owl-features/

[8] T. Berners-Lee, J. Hendler, and O. Lassila, “Scientific american: Feature article: The semantic web: May 2001,” p. 4, (Date accessed: 23.02.2020). [Online]. Available: https://www-sop.inria.fr/acacia/cours/essi2006/Scientific\%20American_\%20Feature\%20Article_\%20The\%20Semantic\%20Web_\%20May\%202001.pdf

[9] I. Oliver, “M3 information SmartSpaces technology overview,” (Date accessed: 05.12.2019). [Online]. Available: https://www.slideshare.net/ianoliver79/m3-information-smartspaces-technology-overview

[10] SOFIA - smart m3 information-sharing platform. NOKIA. (Date accessed: 29.01.2020). [Online]. Available: https://www.slideshare.net/sofiaproject/sofia-smart-m3-informationsharing-platform

[11] “A mechanism for managing and distributing information and queries in a smart space environment:,” in *Proceedings of the Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems*. SciTePress - Science and and Technology Publications, pp. 145–153. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0002193101450153

[12] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, “A survey of information-centric networking,” vol. 50, no. 7, pp. 26–36. [Online]. Available: http://ieeexplore.ieee.org/document/6231276/

[13] K. Yudenok and I. Nikolaevskiy, “Smart-m3 security: Authentification anc authorization mechanisms,” in *2013 13th Conference of Open Innovations Association (FRUCT)*, pp. 153–162, ISSN: 2343-0737.

[14] A. Gurtov, M. Komu, and R. Moskowitz, “Host identity protocol: Identifier/locator split for host mobility and multihoming,” (Date accessed: 15.02.2020). [Online]. Available: https://www.researchgate.net/publication/233893326_Host_identity_protocol_Identifierlocator_split_for_host_mobility_and_multihoming

[15] M. Etelapera, J. Kiljander, and K. Keinanen, “Feasibility evaluation of m3 smart space broker implementations,” in *2011 IEEE/IPSJ International Symposium on Applications and the Internet*, pp. 292–296.

[16] A. Lindgren, F. B. Abdesslem, B. Ahlgren, O. Schelén, and A. M. Malik, “Design choices for the IoT in information-centric networks,” in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 882–888, ISSN: 2331-9860.

[17] J. Honkola, H. Laine, R. Brown, and I. Oliver, “Crossdomain interoperability: A case study,” in *In Smart spaces and*, pp. 22–31.

Clustering with Deep Neural Networks – An Overview of Recent Methods

Janik Schnellbach, Marton Kajo*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: janik.schnellbach@tum.de, kajo@net.in.tum.de

***Abstract*—The application of clustering has always been an important method for problem-solving. As technology advances, in particular the trend of Deep Learning enables new methods of clustering. This paper serves as an overview of recent methods that are based on Deep Neural Networks (DNNs). The approaches are categorized depending on the underlying architecture as well as their intended purpose. The classification highlights and explains the four categories of Feedforward Networks, Autoencoders (AEs) as well as the generative setups of Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). Subsequently, a comparison of the concepts points out the advantages and disadvantages while evaluating their suitability in the area of image clustering.**

***Index Terms*—Deep Neural Networks, Deep Clustering, Variational Autoencoder, Generative Adversarial Net**

Figure 1: John Snow's Death Map [1]

1. Introduction

The basic idea of clustering is the analysis of data with the aim to categorize it into groups sharing certain similarities. The assessed data can range from a small number of characteristics to a huge multidimensional set. Because it is expected to derive certain trends from the input, clustering is a common method to solve practical problems.

A particular example is the application of clustering as performed by John Snow back in the 19th century. John Snow worked as a physician during the cholera epidemic in London. His idea was to mark the cholera deaths on a map of the city, as one can see in Figure (1). Since the deaths notably centered around water pumps, he discovered the correlation between the water supply and the epidemic.

While John Snow did his clustering task manually on a sheet of paper, nowadays methods allow clustering in an automated manner. The application of Artificial Intelligence enables to process big amounts of data while being way more effective. One can distinguish between Supervised and Unsupervised Learning. Supervised Learning assigns the data to prior defined classes of characteristics and qualities. This process is also called classification. In contrast, Unsupervised Learning, of which one category is clustering, can uncover those classes simply from the given set of data without preliminary definitions [2]. The methodology of clustering can either be generative or discriminative. The generative approach tries to work out the data distribution with statistical models such as a Gaussian Mixture Model (GMM) or the k-means algorithm. These models will be explained later in the paper. Discriminative Clustering on the other hand applies separation and classification techniques to map the data into categories without any detour. Regularized information maximization (RIM) is a famous example of this type and will also be discussed in the next section [3].

As both, the amount of data as well as the type of data can vary considerably, a steadily growing selection of methods is currently available. With an increasing amount of approaches, it can be difficult to maintain an overview of the various concepts. The recently published work of Technical University in Munich [4] discusses the current state of the art deep clustering algorithms in a taxonomy. The authors give an overview of the different approaches on a modular basis to provide a starting point for the creation of new methods. However, it lacks proper classification of currently available frameworks, as the authors rather have an eye for the composition of methods instead of the big picture. For this reason, our paper makes a further contribution towards this set of methods with a more detailed description of the concepts as well as a proper classification of them. As it has only been marginally included in the recent paper, special attention is given to novel trends in the area of Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs).

In the following, Section 2 describes the different categories for clustering with Deep Neural Networks (DNNs). For each category, several methods are illustrated. Subsequently, Section 3 does provide an evaluation of the aforementioned methods, with regard to the application area of images, followed by a summary in Section 4.

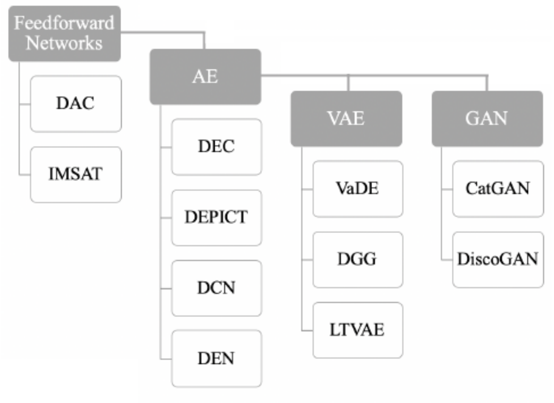


Figure 2: Overview of methods that are addressed in this paper. Feedforward Networks are the basic building block for AEs. VAEs and GANs then again consist of AEs themselves.

2. Deep Clustering

2.1. Feedforward Networks

As a standard setup of a Neural Network, one can define a group of Feedforward Network architectures that follow the same approach: the optimization of a specific clustering loss [5]. This category can be subdivided into Fully-Connected Neural Networks (FCNs) and Convolutional Neural Networks (CNNs).

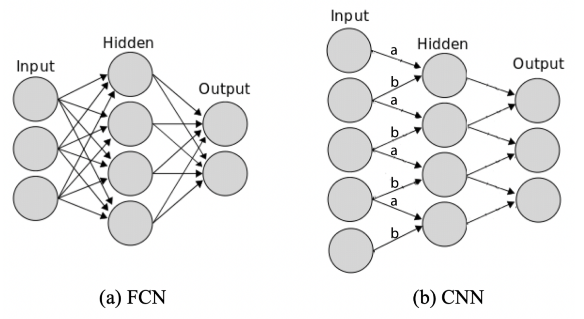


Figure 3: Layout of Feedforward Networks [6]

FCN is also frequently called Multilayer Perceptron (MLP). This architecture has a topology where each neuron of a layer is connected with every neuron on the subjacent layer. The links between neurons have their own weight, regardless of the other connections. CNNs, on the other hand, are rather inspired by the biological layout of neurons, which means that a neuron is only connected to a few others of the overlying layer [5]. In contrast to FCN, a consistent pattern of weightings is used between the neurons of two layers. Figure (3) illustrates the layouts and their weighting described above.

Deep Adaptive Clustering (DAC) is an approach for image clustering, developed by the University of Chinese Academy of Sciences. Due to the area of application, it is also called Deep Adaptive Image Clustering. DAC handles the relationship of two pictures as a binary relationship. By doing this, it decides whether an image matches a certain cluster or not. The pictures are compared by the cosine distance of previously calculated label features, that are extracted from the images by a CNN. Based on the results, the framework decides whether the pictures belong to the same or different clusters. However, this method requires a good initial distribution of clusters, which can be hard to initialize [7].

Information Maximizing Self-Augmented Training (IMSAT) The Previously described feedforward method is based on CNNs. However, this paper seeks to provide a broad overview of the different approaches pending on the network architecture. An example for the application of FCNs is IMSAT. This method is based and advanced from the method of Regularized Information Maximization (RIM) [8].

The basic idea is to handle both the class balance as well as the class separation, meaning that RIM has the objective to balance the amount of data entities inside the clusters. The underlying FCN applies a function that maps data dependent upon the similarity into similar or dissimilar discrete representations. Additionally, Self Augmentation is applied to the data set. This is done, in order to impose the invariance on the data representations [9].

2.2. Autoencoder (AE)

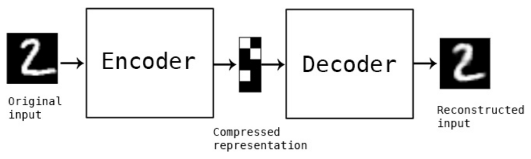


Figure 4: Basic layout of an AE [10]

The above described Feedforward Networks can be used to assemble the network of an AE, which is shown in Figure (4). It consists of an encoder and a decoder [11]. Both have different tasks during their training phase. While the encoder maps the input data according to an encode function within a latent space, the decoder reconstructs the initial input data with the objective of a minimal loss on the reconstruction [12]. The encoder, as well as the decoder, can either be constructed as FCN or CNN. The setup can be trained according to a certain data set [5].

Training can be divided into two phases. While one can separate the two phases in a logical way, both are generally realised simultaneously. During the first phase, the AE performs a pretraining while focusing on the minimization of the basic reconstruction loss. The optimization of this parameter is carried out by any type of AE. The second phase can be seen as a finetuning of the network. The approaches for this step can differ substantially, as various kinds of clustering parameters can be used to optimize the result. The different finetuning strategies are described as part of the approaches presented in the following paragraphs [4].

Deep Embedded Clustering (DEC) is possibly the most significant contribution in the area of clustering with AEs. For the second phase, the so-called cluster assignment hardening loss is optimized. The framework targets to minimize the Kullback–Leibler divergence between an initially computed soft assignment and an auxiliary target distribution. This is done iteratively, with an accompanied improvement of the clustering assignment [13]. It is often used as a starting point, as well as a comparison tool for other approaches [14].

Deep Embedded Regularized Clustering (DEPICT) This approach is based on DEC and is particularly suited for image datasets. It mitigates the risk of reaching degenerative solutions by the addition of a balanced assignment loss [4].

Deep Clustering Network (DCN) extends the previously described AE with the k-means algorithm. The k-means optimization tries to cluster the data around so-called cluster centers to enable an easier representation of the data. DCN optimizes k-means along with the reconstruction loss in the second phase [4].

Deep Embedding Network (DEN) The DEN approach has the objective to improve the clustering towards an effective representation. This is done by an additional locality-preserving loss as well as a group sparsity loss that are jointly optimized in the second phase [14].

2.3. VAEs

While the two aforementioned types can result in high-quality clustering, they are not able to point out the actual coherence of the analyzed data set. Knowledge about that enables to synthesize sample data from the existing dataset. This can be particularly impressive for pictures. In a nutshell, VAE is a refined variant of the traditional AE that forces the AE cluster to impose a certain distribution. It optimizes the lower bound of a data log-likelihood function [15].

Variational Deep Embedding (VaDE) VaDE uses a GMM as the predefined distribution. The GMM selects a fitting cluster that is subsequently transposed towards an observable embedding by a DNN [15].

Deep clustering via GMM VAE with graph embedding (DGG) extends the GMM with stochastic graph embedding in order to address a scattered and complex spread of data points. Graph embedding is applied to the pairs of vertexes in a similarity graph. The objective is to retain information about the relationship of the pairs while mapping each node as a vector with preferably low dimension [16]. The relationship and similarity among pairs are calculated by a minimization of the weighted distance, using their posterior distributions. In summary, DGG optimizes a combination of the loss of the previously described graph embedding with the already known GMM distributive function [17].

Latent tree VAE (LTVAE) has been published by researchers from Hong Kong earlier this year. Their framework takes a particular account of the multidimensionality and the associated range of differentiating structures concerning the data. A tree structure is used, built by multiple latent variables, each including a partition of data. During a learning phase, the tree updates itself, using the relationships among the different facets of the data. Figure (5) shows four different facets as the outcome of clustering applied to the STL-10 dataset. It can be observed that (b) Facet 2 has an emphasis on the front of the cars, compared to the other facets. In general, Facet 2 seems to have a relation to the eyes and lights of the objects. Also, when comparing the deers of facet 2 and 3, one can recognize a pattern in facet 2 with an emphasis on the antlers of the animals [18].

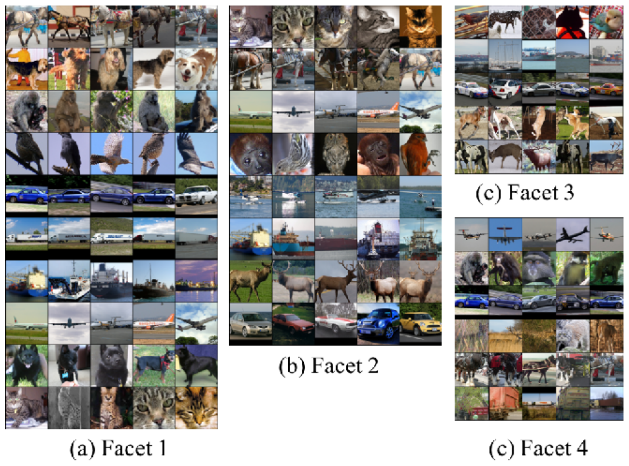


Figure 5: Results for application of LTVAE to STL-10 [19]

2.4. GANs

Next to VAEs, we take a closer look at GANs. A GAN is constructed from a generator and a discriminator. Those two operate in a minimax game. The generator is trained towards a distribution of a certain data set. The discriminator has the task to verify whether a sample from this distribution is a real one or a fake one. Based on this verification, feedback is given to the generator which is used to further improve the sample quality [20].

Categorical GAN (CatGAN) A popular modification of the common GANs are the CatGANs. In simple terms, the discriminator no longer decides whether the samples are real or not. Instead, samples are assigned to appropriate categories. CatGANs use a combination of generative and discriminative approaches. This novel approach requires the generator to spread the samples across the different categories in a balanced way and, most importantly, the generated samples need to be clearly classifiable for the discriminator [3, Section 3.2].

Discover relations between different domains (DiscoGAN) DiscoGANs are based on the idea of cross-domain relations. Human beings are able to understand correlations among different entities. For instance, one can discover the relationship between shoes and handbags that share a resemblance in their color

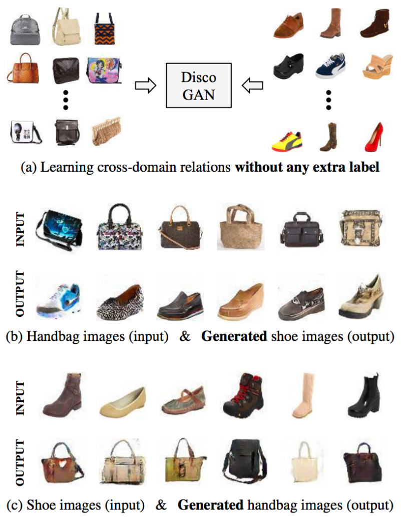

Disco GAN
(a) Learning cross-domain relations **without any extra label**
INPUT
OUTPUT
(b) Handbag images (input) & **Generated** shoe images (output)
INPUT
OUTPUT
(c) Shoe images (input) & **Generated** handbag images (output)

Figure 6: Application of DiscoGAN [22]

sample. Figure (6) presents the application of DiscoGANs on this particular example. Mutually independent image sets of shoes on the one hand and bags, on the other hand, are subject to this picture. Depending on the input, the GAN finds a visually appropriate match.

DiscoGANs can associate an entity from a given pool of entities to a fitting entity from a different pool of entities. This is achieved by coupling two different GANs, which are able to map each entity to the opposite entity [21]. This technique enables to discover links between different clusters and therefore DiscoGANs may create new clusters by combining existing ones.

3. Discussion

After the previous section pointed out the different categories with the different types, this part focuses on the application as well as the advantages and disadvantages of the frameworks. The comparison is made on the level of categories, focusing on the application area of images. Since FCNs are fully connected, they are less suited for image processing. For high-resolution images, FCNs quickly find themselves reaching the limits of feasibility in terms of trainability and depth. Therefore, CNNs are rather suited for images. Depending on the requirements, the depth of Feedforward Networks and in particular of CNNs can be adapted.

The depth of AEs is rather limited since the opposing layout of decoder and encoder requires the depth on both sides. Instead, AEs offer the usage of different clustering parameters, which can be jointly optimized. Conventional Feedforward Networks solely optimize clustering loss.

In contrast to the previous methods, VAEs and GANs feature the ability of sample generation. In general, the optimization process of both can be expected to require a larger extent of computing power than Feedforward Networks and AEs [5]. Considering images once more, GANs usually score better than VAEs in terms of image quality, as the usage of the maximum likelihood approach tends to deliver blurry images. With a more rapid generation and better quality through a generative model, GANs usually score better. It can be said that the general setup allows a more extensive and rather flexible usage in comparison to VAEs [23].

This paper does offer a large extent of recent approaches and methods. In addition, we want to provide further food for thoughts in the area of deep clustering.

Deep Believe Networks (DBNs) As briefly mentioned in the context of DGG, there is a group of generative graphical models that have not been mentioned yet. DBNs are assembled by multiple stacked Restricted Boltzmann machines (RBMs). The starting paper [4] provides Nonparametric Maximum Margin Clustering (NMMC) as an example for DBNs.
Further types of GANs do also apply adversarial nets with the objective of clustering. Information Maximizing Generative Adversarial Nets (InfoGANs) learn the disentangled representation of the data and are particularly suited for scaling of complex datasets [5]. Other types may not have an immediate link to the task of clustering. However, the fundamentals of those might be useful for future research. Stacked GANs (StackGANs), for instance, address the task of image generation based on textual descriptions. It is based on a divide and conquer approach that splits up the problem into smaller subproblems [24].
VAE-GANs combine the two approaches of sample generating methods. As described in [25], the idea is to replace the decoder of a VAE with a GAN. This tries to deal with the blurry images that were mentioned earlier in this section. The idea behind its design is to cope with the VAE's reconstruction task by utilizing the detected feature representation from the discriminator of the GAN. However, as mentioned before, both require much computing power, which applies all the more for a combination as described above.

4. Conclusion

In this paper, we have emphasized the opportunities for clustering, which emerge through the recent advancements in the area of Deep Learning. Based on the network layout we derived different categories. For each of them, several frameworks are described in detail, featuring information about a preferred application area. In addition, we provided a comparison of the categories which included a specific focus on image clustering with special attention to the respective advantages and disadvantages. Finally, we give a further reference to different technologies that haven't been mentioned in this paper.

Overall, our paper has provided a general overview of the existing clustering frameworks and can further be used to get deeper into either the general topic of Deep Clustering or a specific type of category.

References

[1] [Online]. Available: http://blog.rtwilson.com/wp-content/uploads/2012/01/SnowMap_Points-1024x724.png

[2] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, 2013. [Online]. Available: http://dx.doi.org/10.14569/IJARAI.2013.020206

[3] J. T. Springenberg, "Unsupervised and semi-supervised learning with categorical generative adversarial networks," 2015.

[4] E. Aljalbout, V. Golkov, Y. Siddiqui, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *CoRR*, vol. abs/1801.07648, 2018. [Online]. Available: http://arxiv.org/abs/1801.07648

[5] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

[6] [Adjusted]. [Online]. Available: https://www.researchgate.net/profile/Eftim_Zdravevski/publication/327765620/figure/fig3/AS:672852214812688@1537431877977/Fully-connected-neural-network-vs-convolutional-neural-network-with-filter-size-1-2.ppm

[7] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, "Deep adaptive image clustering," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 5880–5888.

[8] A. Krause, P. Perona, and R. G. Gomes, "Discriminative clustering by regularized information maximization," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 775–783. [Online]. Available: http://papers.nips.cc/paper/4154-discriminative-clustering-by-regularized-information-maximization.pdf

[9] W. Hu, T. Miyato, S. Tokui, E. Matsumoto, and M. Sugiyama, "Learning discrete representations via information maximizing self-augmented training," 2017.

[10] [Online]. Available: https://i.stack.imgur.com/zzzp7.jpg

[11] N. Mrabah, N. M. Khan, and R. Ksantini, "Deep clustering with a dynamic autoencoder," *CoRR*, vol. abs/1901.07752, 2019. [Online]. Available: http://arxiv.org/abs/1901.07752

[12] D. Berthelot, C. Raffel, A. Roy, and I. J. Goodfellow, "Understanding and improving interpolation in autoencoders via an adversarial regularizer," *CoRR*, vol. abs/1807.07543, 2018. [Online]. Available: http://arxiv.org/abs/1807.07543

[13] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *CoRR*, vol. abs/1511.06335, 2015. [Online]. Available: http://arxiv.org/abs/1511.06335

[14] T. Yang, G. Arvanitidis, D. Fu, X. Li, and S. Hauberg, "Geodesic clustering in deep generative models," *CoRR*, vol. abs/1809.04747, 2018. [Online]. Available: http://arxiv.org/abs/1809.04747

[15] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: A generative approach to clustering," *CoRR*, vol. abs/1611.05148, 2016. [Online]. Available: http://arxiv.org/abs/1611.05148

[16] S. Yan, D. Xu, B. Zhang, H. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 40–51, Jan 2007.

[17] L. Yang, N.-M. Cheung, J. Li, and J. Fang, "Deep clustering by gaussian mixture variational autoencoders with graph embedding," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[18] X. Li, Z. Chen, and N. L. Zhang, "Latent tree variational autoencoder for joint representation learning and multidimensional clustering," *CoRR*, vol. abs/1803.05206, 2018. [Online]. Available: http://arxiv.org/abs/1803.05206

[19] [Online]. Available: https://d3i71xaburhd42.cloudfront.net/7b85357834e398437a291906aded59caff5151eb/9-Figure6-1.png

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[21] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," *CoRR*, vol. abs/1703.05192, 2017. [Online]. Available: http://arxiv.org/abs/1703.05192

[22] [Online]. Available: https://ieee.nitk.ac.in/blog/assets/img/GAN/discogan.png

[23] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," 2016.

[24] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," *CoRR*, vol. abs/1612.03242, 2016. [Online]. Available: http://arxiv.org/abs/1612.03242

[25] A. B. L. Larsen, S. K. Sønderby, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *CoRR*, vol. abs/1512.09300, 2015. [Online]. Available: http://arxiv.org/abs/1512.09300

Fault tolerance in SDN

Leander Seidlitz, Cora Perner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: leander.seidlitz@tum.de, clperner@net.in.tum.de

***Abstract*—Software Defined Networking (SDN) is based on decoupling the data and control plane of network devices. Switches handle packet forwarding in the data plane. A centralized controller offers a global network view to network applications and enables configuration through a single point. Paths through the network may be configured end-to-end, the centralized controller takes care of configuring the switches.**

While SDN offers high flexibility, fault-tolerance becomes an issue. The global view of the controller allows for fast failover in the data plane. Fault-tolerance in the control plane is a more complex problem. In order to correctly process incoming packets a controller must be available at any time. The fault-tolerance of the control layer is vital for the function of the network

This paper gives an overview of current approaches to fault-tolerance in the data as well as control plane.

***Index Terms*—software-defined networking, fault-tolerance**

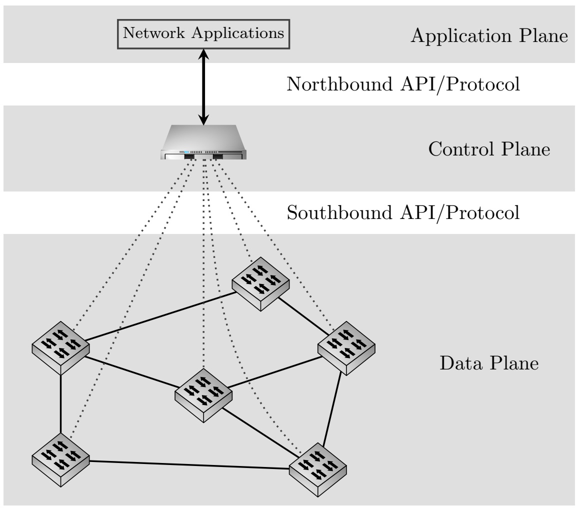


Figure 1: Structure of a SDN network.

1. Introduction

Software Defined Networking (SDN) offers greater flexibility than traditional network architectures. In traditional network topologies, control and data plane are distributed over the network. Routing protocols such as OSPF (Open Shortest Path First) [1] are used by the distributed entities in order to establish routes through a network. Network applications have to communicate with multiple devices in order to configurate the network.

In contrast, SDN splits up the data, control and application plane of a network, as depicted by Figure 1. While the data plane is responsible for packet filtering and forwarding, the control plane enforces policies and handles tasks as load balancing and multipath routing. A central controller configures the data plane by sending commands to the respective switches. It offers an abstracted view of the network topology and flows to the network applications in the application plane, enabling network configuration through a single controller. While the data and control plane take care of handling flows in the network, the application plane manages network policies using the global network view presented by the control layer.

The switches in the data plane rely on a Network Information Base (NIB) in order to handle packets. Packets arriving at the ingress port of a switch are matched to flows. The NIB specifies how to handle the packet, and whether to forward or drop it. Packets belonging to an unknown flow are forwarded to the controller. The controller decides how to handle the packet, for example by configuring a flow through commands to the respective switches. The flow configuration received by the switches through controller commands is saved in the NIB. Future packets of the established flow can be handled without consulting the controller.

Communication between the control plane and application plane is done through a northbound Application programming interface (API) or protocol. As of December 2019, there are no standarized APIs or protocols for the northbound interface, although Representational State Transfer (REST) APIs are widely used.

The control plane communicates with the switches through a southbound API or protocol. OpenFlow [2] is a common protocol for the communication between control and data plane.

While the separation of data and control plane offers flexibility, the centralized control plane creates a single point of failure. To ensure correct network function, the control plane must be available at any time.

The Fault Tolerance Problem. Networks are expected to operate without disruption, even in the presence of link or device failures. Faults in the network should be handled quickly and transparently, causing only minimal service interruption. The strict separation of data and control plane forces us to handle fault-tolerance for both planes separately. SDNs require approaches to fault-tolerance in two domains: The data plane, where switches or links can fail as well as the control plane, where controllers or the link between controller and switch may fail.

We will focus on fault-tolerance in the control plane. Section 2 gives an overview of fault-tolerance in the data plane. Approaches to fault-tolerance in the control plane are discussed in Section 3. Fault-tolerance in the application plane is not in the scope of this paper.

2. Fault Tolerance in the Data Plane

The data plane takes care of handling packet flows in the network. Flows are established by the controller through configuration of the individual switches. For reliable network operation the data plane must be resilent against link and switch failures. Failures have to be detected quickly and resolved by rerouting affected traffic on alternative links, restoring the networks functionality.

Still, basic network policies must not be violated. For example, traffic rerouted on a different path through the network should not be able to bypass a firewall. Fault-tolerance mechanisms therefore do not only have to regard the network topology but also its policies configured by the application layer.

The controller possesses global knowledge of the network topology and can therefore run centralized algorithms. These are potentially more efficient than distributed algorithms, such as the *rapid spanning tree protocol* [3], which only have limited information about the network.

Approaches to focusing on data plane fault tolerance, such as FatTire [4], the approach of Paris et al. [5] or CORONET [6], have to ensure resilence of the data plane against failures without introducing large overhead.

2.1. Reacting to Topology Changes

The structure of SDN based networks is not static. Links in the network are removed and established, constantly changing the networks topology. While traffic should usually take a near optimal path through the network, this path may fail. An optimal failover algorithm would choose the next optimal path, but calculating this path can be expensive and therefore time-consuming for large networks. Restoring the network function by applying a suboptimal path outweighs path optimization and is acceptable. A suboptimal path can be optimized after network function is restored.

Traffic traversing the network on an suboptimal path causes overhead. While rerouting the traffic to a lower-overhead path may lower the costs of traversing the network by finding a better path. Reconfiguring the network introduces overhead as well. Approaches to fault-tolerance should only change existing paths if the benefit of rerouting traffic is larger than the overhead caused by reconfiguring.

2.2. Minimizing Overhead

Paris et al. [5] present an approach that finds a balance between optimal paths and the frequency of reconfiguration. They divide their approach into two sub-mechanisms: Firstly, rapid handling of failures by rerouting to alternative paths and secondly a mechanism for path optimization.

Restoring Paths. After a link or device failure backup paths are calculated on demand. The priority is to quickly find an alternative path, which is allowed to be suboptimal. The path is calculated based on a shortest-path algorithm. Restoring the path is vital for the networks function, the path optimization is taken care of by another mechanism.

Optimizing Paths. Optimization of network paths is done by a mechanism Paris et al. call *Garbage Collection of network resources*. Periodically flow allocations in the network are analyzed and optimized. An iterative algorithm converging to the optimal solution is used. As new links become available and failed links are repaired, the garbage collection may reroute traffic, should network changes open up shorter paths. Rerouting is only done in case that the optimization is larger than the overhead caused by the necessary network reconfiguration.

In a static network the paths would converge to the optimal solution. Failed links and devices introduce suboptimal paths and therefore overhead, moving further away from the optimal solution.

2.3. OpenFlow Action Buckets

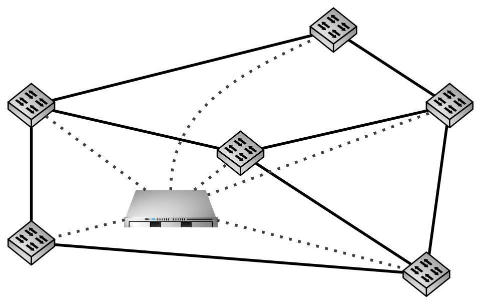
OpenFlow 1.3 [7] introduced the concept of action buckets. An action bucket groups a number of rules, the bucket itself is bound to conditions based on switch state, such as the status of a link.

Action buckets allow creating conditional forwards such as deactivating a set of rules as a link fails. The buckets are prioritized. Packets are matched with the rules in the highest bucket which conditions are met. This allows specifying precomputed backup paths that become instantly active when a link fails. FatTire [4] makes use of OpenFlow action buckets.

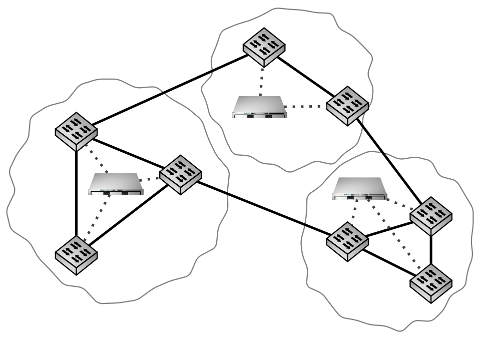
The FatTire Language allows the definition of network paths as regex-like expressions. Paths in the network are specified end-to-end, the necessary degree of fault tolerance can be specified for each path. The FatTire compiler then calculates the hops through the network for path realization as well as possible backup paths. The result is an OpenFlow configuration that can be applied to the individual switches. As links fail the precomputed backup paths become active. Repaired links are instantly reused.

2.4. Fault-Tolerant Controllers

The solutions presented above depend on the controller being available at any point. Nevertheless, controller failures are possible and must be handled. A failed control plane leaves the network in a headless state. Events such as incoming packets belonging to unknown flows cannot be handled without a controller. Therefore, a fault-tolerant control plane is vital. In the following section we will present approaches to a fault-tolerant control plane.



(a) Centralized



(b) Distributed

Figure 2: SDN Control Plane Topologies.

3. Fault-Tolerance in the Control Plane

While fault-tolerance in the data plane is essential, the data plane is dependent on the controller. The control plane is vital for the operation of the network as it handles tasks such as deciding how to handle unknown flows as well as the communication with the application layer. The controller must be operational at all times to ensure correct network operation. SDN builds on a centralized controller. This controller presents a single point of failure. Introducing redundancy to the control plane allows for a fast failover in case of a controller fault. In the following we present approaches to fault-tolerant controllers.

3.1. Control Plane Topology

SDN control planes either have a single logically centralized controller (Figure 2a) or are constructed in a distributed topology (Figure 2b). A distributed topology has multiple controllers, each handling a separate domain of a network. Distributed typologies are mostly found in large networks in which a single controller cannot handle the load. Multiple parallel operating controllers offer the possibility of one controller taking over another part of a network as its controller fails.

The solution to fault-tolerance in a distributed controller scheme can be found in approaches such as HyperFlow [8] or ECFT [9]. In the following we will focus on more traditional SDN based networks with a centralized controller.

Logically Centralized Control Planes. In a logically centralized control plane only one controller is active at any time. This controller takes charge of all decisions in the network such as configuring the switches. Fault-tolerance in a logically centralized scheme is commonly achieved through a master-slave approach.

A master controller operates the control plane, slave controllers passively mirror the master controllers state. As the master controller fails, one of the slave controllers becomes master and takes over.

The failover time is critical. The network is inpoerable as long as the controller is not available.

3.2. Ravana: A Master-Slave Approach

In the following we discuss master-slave approaches to fault-tolerance in the control plane on the example Ravana [10]. In a master-slave topology, multiple controllers are present. One controller serves as master controller, controlling the network. Slave controllers mirror the master, and take over if the master fails.

Ravana [10] is an approach ensuring fault-tolerance for the controller, the communication between controller and backup controllers as well as for the communication between switches and the controller. It was proposed by Katta et al. A solution to fault-tolerance must fulfill the following requirements:

A) Total event ordering
B) Exactly-once event processing
C) Exactly-once execution of commands
D) Consistency under switch and controller failures

In the following we analyze Ravana in regard to these requirements.

A) Total event ordering. In a master-slave approach the slaves must have the same view of the network as the master. Each replica of the master builds its state independently, based on the stream of events received. In order to keep the state equal over all replicas the order of events processed must be the same for all controllers. Inconsistent event ordering may be caused by different latencies between switches and controllers.

There are two approaches to keeping the master and its replicas in sync:

1) Let the switch broadcast events to all controllers
2) Replicate the event at the master controller before processing it

Regarding to the first approach, ensuring the same order of events at each controller is challenging. A total order would require the controllers to synchronize the events received, posing a large overhead.

Replicating the event at the master controller before processing ensures the order of events received. It is the same for all controller replicas as the master controller defines the order. In Ravana switches send events to only the current master controller. The switches use an event buffer to prevent lost messages in case of a master controller failure.

By performing event replication at the master controller Ravana ensures a total event order at all replicas.

B) Exactly-once event processing. An approach to fault-tolerance must ensure that every event sent by a switch is processed exactly once. Events must not be lost nor processed repeatedly. The delivery of messages can be ensured by sender-side buffers, repeating the transmit if the receiver has not acknowledged the message.

If a packet causes an event the switch sends the event to the current master controller and additionally saves the event in a local buffer. Events received by the master are replicated to the slave controllers before the master processes them. The slaves hold back the event from their application layer until the master has successfully processed the event. After successful processing of the event, a confirmation message is sent to the switches and slaves. The switch clears the event from its buffer, the slaves can now safely release the event to the application. If the master fails during event processing, the switch can retransmit the event from its event buffer to the new master.

Unique messages IDs and receiver side filtering guarantee that messages are processed at most once.

C) Exactly-once Execution of Commands. As commands from the controller may not be idempotent, we must ensure that they are executed exactly once by a switch. A command buffer at the controller and acknowledgments by the switches, analogous to the switch-side event buffer, ensure that a switch receives and successfully executes a command.

The controller buffers commands sent to the switch, and deletes them from the buffer when the switch acknowledges execution of the command. The replicas of the controller are informed about the command buffer and the status of the commands sent. In case of the master controller crashing after sending a command but before processing the acknowledgment by the switch, the new master controller will find an incomplete command execution in the command buffer. It will resend the command. As commands have unique IDs, the switch will filter the command (ensuring at-most-once execution) but resend the acknowledge to the new master. The master controller will mark the command as successfully executed and replicas are informed.

D) Consistency under Switch and Controller Failures. Switches retransmit events and acknowledge commands. As long as the controller does not fail, we can handle switch failures the same way as single-controller SDN do: Relay the decision to the control application in the application layer. The network application will then decide how to reroute traffic and send appropriate commands to the controller.

Combined switch and controller failures pose a more complex problem. Should the master controller fail before finishing the processing of an event, a slave controller will take over. A switch that has sent an event has therefore not received an acknowledge yet. If this switch fails during this failover, it cannot retransmit the event to the new master. Still, the new master controller has received a copy of the event from the old master as events are replicated before processing. It sees the unfinished event in his buffer and can process it, even without the switch retransmitting. After this, we handle the switch failure as regular switch failure.

We conclude that Ravana solves the requirements needed for reliable fault-tolerance in the control plane. We now discuss how the failed controller is replaced in distributed and centralized topologies.

3.3. Controller Failover

In a distributed control plane topology a controller is responsible for a set of switches. HyperFlow [8] or ECFT [9] are approaches to fault-tolerance in distributed schemes. Both approaches use a similar approach to replacing the failed controller. The switches the failed controller was responsible for are split up and assigned to other controllers. This is done based on metrics such as the delay between the respective switch and controller as well as the controller load. In order to prevent cascading failures controllers must not be overloaded.

In a master-slave topology the slave controllers have to decide on who becomes the new master. Ravana [10] solves this problem by letting the switches contend for a distributed Zookeeper [11] lock. The controller that obtains the lock becomes the new master. The new master then informs the switches of the change in master controller.

3.4. Interfacing with the Application Layer

In a traditional SDN topology, the application layer interfaces with a single controller. A fault-tolerant approach should be observational indistinguishable from a single controller SDN: the system should behave in the same way as a fault-free single controller system would.

Additionally, controller redundancy and failover should be transparent for the application layer. This enables network applications to interface with fault-tolerant control layers without the need of rewriting.

4. Conclusion and Future Work

While fault-tolerance in the data plane seems mostly to be an optimization problem, fault-tolerance in the control plane is a more difficult problem to solve. Concerning the data plane, the global knowledge of the controller allows fast re-routing of traffic in case of failed links and switches. In the control plane, failures are more difficult to handle.

We presented requirements, solutions to fault-tolerance in the control plane must meet, and how approaches can fullfil them. Ravana [10] is a promising master-slave approach to fault-tolerance in the control plane. It offers transparent fault-tolerance and fast failover between controllers. Ravana does require extension of the OpenFlow protocol, which may hinder its acceptance.

Future extensions to master-slave schemes for fault-tolerance in the control plane may base on Ravana and extend it. Current protocols fail at state-replicating multi-threaded control applications as well as handling byzantine faults. These are tasks to be solved by future approaches to fault-tolerance in SDNs.

References

[1] J. Moy, "OSPF Version 2," RFC Editor, RFC 2328, Apr. 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, Mar. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1355734.1355746

[3] IEEE Standards Association, "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks," IEEE, Tech. Rep., May 2018. [Online]. Available: http://ieeexplore.ieee.org/document/6991462/

[4] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*. Hong Kong, China: ACM Press, 2013, p. 109. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491185.2491187

[5] S. Paris, G. S. Paschos, and J. Leguay, "Dynamic control for failure recovery and flow reconfiguration in SDN," in *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN)*. Paris: IEEE, Mar. 2016, pp. 152–159. [Online]. Available: http://ieeexplore.ieee.org/document/7470850/

[6] Hyojoon Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "CORONET: Fault tolerance for Software Defined Networks," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*. Austin, TX, USA: IEEE, Oct. 2012, pp. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/6459938/

[7] Open Networking Foundation, "OpenFlow Switch Specification," Open Networking Foundation, Tech. Rep. Version 1.3, Jun. 2012. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[8] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking*, Apr. 2010, p. 6.

[9] W. H. F. Aly and A. M. A. Al-anazi, "Enhanced Controller Fault Tolerant (ECFT) model for Software Defined Networking," in *2018 Fifth International Conference on Software Defined Systems (SDS)*. Barcelona: IEEE, Apr. 2018, pp. 217–222. [Online]. Available: https://ieeexplore.ieee.org/document/8370446/

[10] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*. Santa Clara, California: ACM Press, 2015, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2774993.2774996

[11] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," *Proceedings of the 2010 USENIX Annual Technical Conference*, p. 14, Jun. 2010.

Time Synchronization in Time-Sensitive Networking

Stefan Waldhauser, Benedikt Jaeger*, Max Helm*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
E-Mail: stefan.waldhauser@tum.de, jaeger@net.in.tum.de, helm@net.in.tum.de

Abstract—**Time-Sensitive Networking (TSN) is an update to the existing Institute of Electrical and Electronics Engineers (IEEE) Ethernet standard to meet real-time requirements of modern test, measurement, and control systems. TSN uses the Precision Time Protocol (PTP) to synchronize the device clocks in the system to a reference time. This common sense of time is fundamental to the working of TSN. This paper presents the principles and operation of PTP and compares it to the Network Time Protocol (NTP).**

Index Terms—**clock, network time protocol (NTP), precision time protocol (PTP), synchronization, time, time-sensitive networking (TSN)**

1. Introduction

In real-time systems, the correctness of a task does not only rely on the logical correctness of its result but also that the result meets some deadline [1]. A typical example of a real-time system is a control system in industrial automation that has to integrate multiple sensor readings and then initiate an action in response. This requires a deterministic underlying network.

Time-Sensitive Networking (TSN) is a set of standards that represent an ongoing effort of the Institute of Electrical and Electronics Engineers (IEEE) 802.1 TSN Task Group [2] to extend and adapt existing Ethernet standards on the data link layer to meet real-time requirements of modern test, measurement and control systems. Advantages over traditional Ethernet include guaranteed bounded latency for time-critical data while transmitting time-critical data and best-effort data on the same network [3].

The foundation of TSN is establishing a shared sense of time in all networked devices in the system. Only then they are able to perform time-sensitive tasks in unison at the right point in time. This shared sense of time is achieved through the use of the Precision Time Protocol (PTP) [3]. PTP is a message based time transfer protocol that enables clocks in the nodes of a packet-based network to synchronize (phase and absolute time) and syntonize (frequency) with sub-microsecond accuracy to a reference time source. PTP was first described in the IEEE 1588-2002 standard. The standard was later revised in IEEE 1588-2008 [4]. The revised protocol is commonly referred to as PTP Version 2 and is used in TSN together with the profiles IEEE 802.1AS and IEEE 802.1ASRev [5].

The paper starts with a description of PTP Version 2 in Chapter 2. The chapter begins with a brief overview of the two fundamental phases of the protocol: the best master clock algorithm and message based time synchronization. Then the various PTP device types and message types are discussed. The two phases are described in more detail in the rest of the chapter.

Next, in Chapter 3, PTP Version 2 is compared to another message based time synchronization protocol called Network Time Protocol (NTP) Version 4. Various advantages and disadvantages are discussed.

Finally, Chapter 4 concludes the paper.

2. Precision Time Protocol

An IEEE 1588 system is a distributed network of PTP enabled devices and possibly non-PTP devices. Non-PTP devices mainly include conventional switches and routers. An example PTP network can be seen in Fig. 1.

The operation of PTP can be conceptually divided into a two-stage process [6]. In the first stage, the PTP devices self-organize logically into a synchronization hierarchy tree using the Best Master Clock Algorithm (BMCA). The devices are continuously exchanging quality properties of their internal clock with each other. The PTP device with the highest quality clock in the system eventually assumes the role of grandmaster (GM) and provides the reference time for the whole system. The subnet scope in which all clocks synchronize to the GM is called PTP domain. The BMCA is explained further in Section 2.3.

In the second stage, time information continuously flows downstream from the GM between pairs of PTP ports with one port in the master state serving time information and the other in the slave state receiving time information. Eventually, the system reaches an equilibrium where all clocks are synchronized to the GM of the system. Time synchronization between master and slave is initiated by the master port, which periodically sends synchronization messages to its slave. These messages are timestamped by the master at transmission and by the slave at arrival. A slave now has two timestamps, the sending time according to the clock of the master, and the receiving time according to its clock. As the message takes some time to travel through the network, the slave also needs to know the network delay to calculate the offset to the master [6].

PTP supports two mechanisms to calculate this delay: End-to-End (E2E) and Peer-to-Peer (P2P). The E2E mechanism requires the slave to measure the total delay between itself and the master (thus end-to-end). The P2P mechanism, on the other hand, requires each device (including switches and routers) on the path between master and slave to measure the delay between itself and its direct neighbor (peer). The total network delay between master and slave is the sum of the peer delays along the path. Technically, E2E can be used in the same domain as P2P as long as the two are not mixed along the same messaging path. Thus, between master and slave, all nodes must either use E2E or P2P [7]. The two mechanisms are discussed in more detail in Section 2.5.

2.1. PTP Device Types

The standard defines five PTP device types: ordinary clocks, boundary clocks, end-to-end transparent clocks, peer-to-peer transparent clocks, and management nodes [4].

An ordinary clock (OC) is an end-device (as opposed to a switch or router) with a single PTP capable port and an internal local clock. It can either assume the role of slave (leave node) or GM (root node) in the synchronization hierarchy.

The main source of error in PTP is asymmetry in the network delay between master and slave. Asymmetric network delay means that sending a message from master to slave takes a different amount of time than the other way around. The most significant sources of asymmetric network delay are different processing and queueing delays in ordinary switches and routers, different data transmission speeds, error differences in the generation of timestamps, and messages taking different routes through the network [6].

IEEE 1588-2008 defines two types of PTP enabled switches and routers to deal with the asymmetry problem: Boundary clocks (BC) and transparent clocks (TC).

A BC has multiple PTP capable ports and one internal clock shared by all ports. If the BC is selected as the GM of the system, then all ports switch to the master state. Otherwise, the BC selects the best clock seen by all of its ports. The corresponding port then switches to the slave state, allowing the internal clock to synchronize. The other ports switch to the master state, serving time information based on the now synchronized internal clock. By terminating and then restarting the time distribution, each BC creates a branch point (internal node) in the synchronization tree. This allows the BC to effectively remove the adverse effects of its processing and queuing delays.

Like a BC, a TC has multiple PTP capable ports, and one shared internal clock. Eliminating asymmetry is achieved by timestamping the entrance and exit of PTP messages that pass through the device. The time the message spent inside the device, called residence time, is calculated by subtracting the entrance timestamp from the exit timestamp. The TC then adds the residence time to a correction field in the PTP message before passing it along. The slave can then remove the accumulated queuing and processing delays by using the correction field value in the offset calculation.

Transparent clocks exist in variants supporting either the P2P delay mechanism or the E2E delay mechanism. Only a single delay mechanism is allowed in the link between master and slave. A boundary clock with ports supporting each of the two mechanisms can be used to connect regions using the different mechanisms. See 1 for an example.

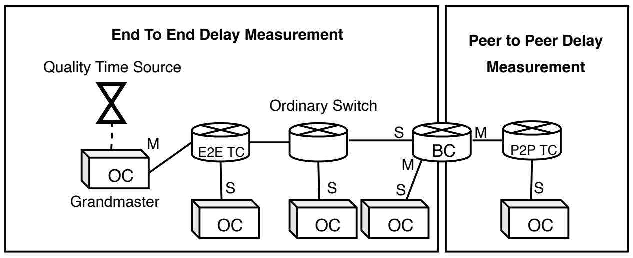


Figure 1: Example PTP Domain (Adapted from [8])

Management Nodes do not take part in the time synchronization but can be used to read and write various PTP properties of other nodes via *Management* messages [6].

2.2. PTP Message Types

IEEE 1588 defines two groups of PTP messages [4]: (1) Event messages, which require an accurate timestamp both at sending and receiving because PTP uses these as timing events. (2) General messages, which are being used to transmit information. In contrast to event messages, sending and receiving of general messages does not produce a timestamp.

The event message are *Sync*, *Delay_Req*, *Pdelay_Req* and *Pdelay_Resp*. These are used in the time synchronization process to transfer timestamps and correction information between master and slave. The general messages are *Announce*, *Follow_Up*, *Delay_Resp*, *Pdelay_Resp_Follow_Up*, *Management* and *Signaling*. *Announce* messages are used in the BMCA to exchange clock quality information. *Management* messages are used to configure PTP devices. *Signaling* messages are used by PTP clocks to communicate in special settings, such as unicast environments.

IEEE 1588 clocks can either support the one-step or two-step messaging mechanism. When sending *Sync* or *Pdelay_Resp* messages, clocks need to tell the receiver the sending timestamp. They are either capable of including this timestamp in the *Sync* and *Pdelay_Resp* themselves (one-step-clock), or they need to send a second follow-up message containing it (two-step-clock). *Follow_Up* and *Pdelay_Resp_Follow_Up* messages are used for this [9].

As mentioned before, any delay asymmetry causes a loss in accuracy. ‘Artificial’ network delay is created if the timestamps that are generated on the path from master to slave have a different error than those generated on the path from slave to master. This happens for example if software timestamping is used because the operating system and protocol stack packet processing delay fluctuates. Therefore it is recommended to use devices with PTP enabled NICs (Network Interface Cards) in the network. These specialized NICs have a clock, which is used to timestamp the received and transmitted PTP messages as close to the physical layer as possible [10]. Timestamps generated via hardware support have a constant low error and therefore improve synchronization accuracy.

PTP usually is implemented using multicast communication, but it can also be configured for unicast messaging. The PTP standard does not require any specific transport protocol, but most commonly, UDP is used. The well known UDP ports for PTP traffic are 319 (Event Message) and 320 (General Message) [11].

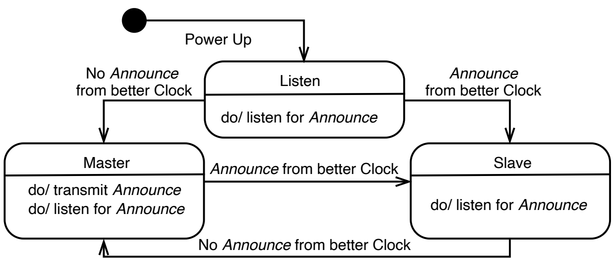

Power Up
No *Announce* from better Clock
Listen
do/ listen for *Announce*
Announce from better Clock
Master
do/ transmit *Announce*
do/ listen for *Announce*
Announce from better Clock
Slave
do/ listen for *Announce*
No *Announce* from better Clock

Figure 2: Simplified PTP State Machine of an Ordinary Clock (Adapted from [12])

2.3. Best Master Clock Algorithm

IEEE 1588 is an administration-free system that can deal with events like system restarts, failure of a clock, or changes in network topology automatically. This is achieved via the BMCA, which runs continuously in OCs and BCs in a domain [4].

The basics of the BMCA can be explained using the simplified state diagram of an OC in Fig. 2. All OCs listen for defined intervals to *Announce* messages, which are sent in a specific frequency by ports in the master state to the PTP multicast address. These messages contain attributes about the sending clock. At the end of each listening interval, an OC has either received an *Announce* message from a better clock or not.

The attribute comparison algorithm uses the following criteria in order of precedence to determine if an *Announce* message from a better clock has been received [12] [4]:

1) *priority1*: This is a user configurable field. It is the first parameter to be considered by the BMCA. Therefore an administrator can manually set up a clock quality hierarchy.
2) *clockClass*: This field generally described the quality of a clock. A clock connected to a GPS receiver has a higher class than a free-running clock.
3) *clockAccuracy*: This field describes the accuracy of the clock. The value is picked from defined accuracy levels in the standard, for example, 25 ns to 100 ns.
4) *offsetScaledLogVariance*: This field describes the stability of the clocks oscillator.
5) *priority2*: This is a user configurable field. It can be used to manually rank clocks of equal quality.
6) *clockIdentity*: This field is usually set to the Ethernet MAC address. It is a unique number that is used to break ties.

If a message from a better clock has been received, a master OC switches to the slave state. If no such message has been received, a slave OC switches to the master state and starts transmitting *Announce* messages. Freshly rebooted OCs are in a special listening state and can either switch to the master or slave state [12].

The process in BCs is similar, but these devices have to compare all of the *Announce* messages received on all the ports, to determine if they become a GM (all ports in master state) or just a branching point (one port in slave state and the others in master state) [12].

Eventually, only a single clock assumes the role of GM in the domain.

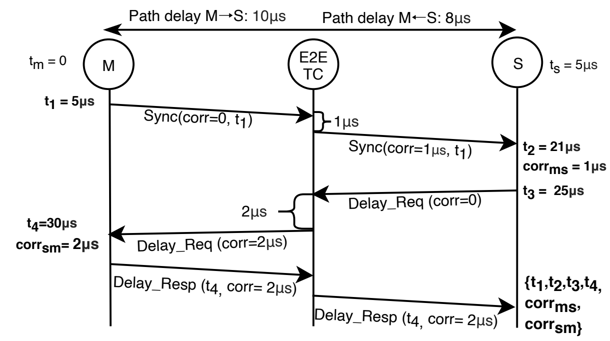


Figure 3: E2E Synchronization (One-Step-Clocks) (Adapted from [4])

It is important to note that the BMCA never stops running. This allows the system to react to certain events by dynamically changing the synchronization hierarchy. For example, if the current GM gets disconnected from the network, a new GM is determined automatically.

2.4. Syntonization

Syntonization in this context means frequency locking two clocks by agreeing on the length of a second. Syntonized clocks, therefore, are running at the same rate. This paper does not discuss the details of syntonization using PTP, but it is important to note that any port in the slave state and any TC syntonizes to the GM [4].

2.5. Synchronization

Time synchronization implies phase-locking two clocks and making them agree on the same time of day. Phase locking means that incrementing the time does not only happen at the same rate in both clocks but also at the same time. Agreeing on the time of day means synchronizing the 'wristwatch time' – year, month, day, hour, minute, seconds and so on in a given timezone. Any OC or BC with a port in the slave state synchronizes to its respective master in the hierarchy [4].

2.5.1. E2E Synchronization. Fig. 3 shows the message exchange to synchronize a one-step slave clock and a one-step master clock with an E2E transparent clock between them. In the example, there exist two sources of delay asymmetry: (1) A difference of 1 µs in the TC processing time. The negative effects of this asymmetry can be automatically removed. (2) A difference of 2 µs that has its origin in transmission speed or path length differences. PTP can not automatically remove the influence of this asymmetry. However, if measured manually, PTP can be configured to account for it [6].

As seen in the example, the slave collects four timestamps and two correction values during the message exchange:

- t_1: *Sync* sending timestamp in master time. In a two-step clock this timestamp is contained in a separate *Follow_Up* message and not in the *Sync* message itself.
- $corr_{ms}$, $corr_{sm}$: Each TC on the path adds the residence time to the correction field in the *Sync* or *Delay_Req* message

- t_2: *Sync* receiving timestamp in slave time
- t_3: *Delay_Req* sending timestamp in slave time
- t_4: *Delay_Req* receiving timestamp in master time

The fundamental assumption of all synchronization protocols that are based on the exchange of timing information via networks with unknown propagation delays is a symmetric network delay between master and slave [6].

Under this assumption the slave is able to calculate the network delay d between itself and the master by dividing the corrected round-trip delay by two:

$$d = \frac{[(t_4 - t_1) - (t_3 - t_2)] - corr_{ms} - corr_{sm}}{2} \quad (1)$$

This assumption is critical since it is not possible to determine one-way delays with an unknown clock offset.

In the example:

$$d = \frac{(30\,\mu s - 5\,\mu s) - (25\,\mu s - 21\,\mu s) - 1\,\mu s - 2\,\mu s}{2} = 9\,\mu s$$

The slave can now calculate the offset o from the master by subtracting from t_2 (slave time): t_1 (master time), the network delay, and the TC correction factor. The result represents the part of the timestamp difference that originates from the slave and master clock divergence.

$$o = t_2 - t_1 - d - corr_{ms} = 21\,\mu s - 5\,\mu s - 9\,\mu s - 1\,\mu s = 6\,\mu s \quad (2)$$

The actual offset is 5 µs, so there is an error of 1 µs. This error occurs because the above assumption was wrong: There is uncorrected asymmetry in the delay between master and slave of 2 µs. However, the example demonstrated that PTP is successfully able to remove the amount of asymmetry stemming from queue effects in ordinary switches and routers by replacing them with TCs.

In general for the error e:

$$e = \frac{ND_{ms} - ND_{sm}}{2} = \frac{10\,\mu s - 8\,\mu s}{2} = 1\,\mu s \quad (3)$$

The maximum possible error due to asymmetry in the network is, therefore, half of the round-trip delay.

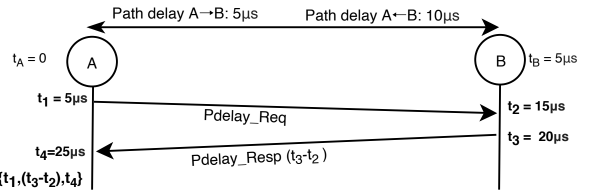


Figure 4: P2P Delay Measurement (One-Step-Clocks) (Adapted from [4])

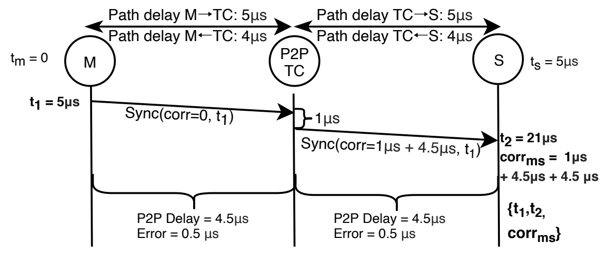


Figure 5: P2P Synchronization (Adapted from [4])

2.5.2. P2P Synchronization. A link between master and slave that is set up to use P2P synchronization calculates the network delay differently. Periodically two directly connected clocks independent of their state perform a message exchange to measure the network delay between them. An example is shown in Fig. 4.

Four timestamps are generated that are used to calculate the network delay:

$$\begin{aligned} d &= \frac{[(t_4 - t_1) - (t_3 - t_2)]}{2} \quad (4)\\ &= \frac{(25\,\mu s - 5\,\mu s) - (20\,\mu s - 15\,\mu s)}{2} = 7.5\,\mu s \end{aligned}$$

The error is again half of the network delay asymmetry between clock A and clock B. This peer delay is measured for both directions. This is important because during the lifetime of the system, the master-slave states of A and B can change.

Fig. 5 shows an example of time synchronization between master and slave using the P2P mechanism with the same delay values as in the E2E case. The timestamps t_1 and t_2 are still created by sending a *Sync* message from master to slave, but the network delay is calculated differently.

Each clock on the link that receives the *Sync* message adds the peer delay value to the correction field. In addition, the TCs add the residence time to the correction field as usual. The correction field, therefore, always represents the network delay from the master until the current node.

The slave adds the final peer delay to the correction field and can now calculate the offset to the master:

$$\begin{aligned} o &= (t_2 - t_1) - corr_{ms} \quad (5)\\ &= (21\,\mu s - 5\,\mu s) - 10\,\mu s = 6\,\mu s \end{aligned}$$

The error is the same as in the E2E example because the total error is just the sum of all errors made during the peer network delay calculation.

Even though no higher precision can be achieved using the P2P mechanism, there are several other factors to consider [7]:

- Ordinary switches and routers do not respond correctly to *Pdelay_Req* messages, in case such devices are used in the network, the E2E mechanism has to be used.
- As the master does only need to respond to *Pdelay_Req* messages from its direct neighbors and not to *Delay_Req* messages from all the slaves that sync to it, a P2P system scales much better. The load on a master that a lot of slaves sync to is dramatically reduced.
- As no *Delay_Req* messages are used, there is no risk of the *Sync* and *Delay_Req* message taking different paths in the network. Thus the risk for delay asymmetry is reduced.

3. Related Work

PTP was designed for usage in local industrial automation and measurement networks where specialized devices like BCs and TCs can be used as switches and routers. Another protocol called Network Time Protocol (NTP), on the other hand, is the workhorse for synchronizing system clocks of devices over the Internet to a common timebase

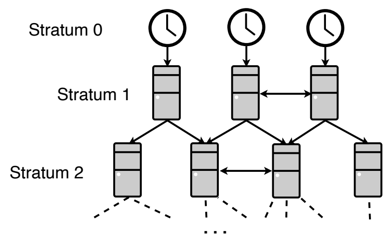


Figure 6: Example NTP Hierarchy

(usually UTC). NTP time synchronization is used, for example, in general-purpose workstations and servers. It is one of the oldest (the first version was released in 1985) protocols still in use today and is currently in its fourth major version. NTP uses UDP on port 123 [13].

Similar to PTP, an NTP network (for example, the global Internet) is hierarchically organized into primary servers, which are directly connected to a reference clock, secondary servers, and clients. In NTP, there also exists the concept of a stratum which represents the logical distance of a server/client to a reference clock. Primary servers have a stratum value of 1 and secondary servers values between 2 and 15. If a server has a stratum value of 16, it means that it is not yet synchronized. A server in stratum n is a synchronization client to a server in stratum $n-1$. In real-world configurations, stratum levels above 4 are rare [14]. Fig. 6 illustrates the hierarchical strata model of NTP. To increase robustness, two NTP servers in the same stratum can also synchronize with each other as peers. If a server loses connectivity to its upstream NTP server, it can receive time information from its peers.

Time synchronization of an NTP client is established through periodic request/reply exchange with one or more NTP servers they are authorized to access. As in PTP, the offset of the client clock to the server clock is calculated from the four timestamps generated during the exchange. The critical source of error is again the delay asymmetry between the two messaging directions.

A key advantage of NTP over PTP is that in NTP, a client polls typically many servers for time synchronization. In case of disagreements between the sources, the most extensive collection of agreeing servers is used to produce a combined reference time, thereby declaring other servers as faulty or not trustworthy [15]. PTP slaves, on the other hand, are trusting a single time source blindly [16]. Slaves can only assume that the calculated offset to the master is correct, as they are not capable of comparing it to some value from other sources. This means that if the GM has some error that causes it so send the wrong time information in *Sync* messages but that does not affect the clock quality presented in *Announce* messages, slaves change their clock to the wrong time. Several researchers have proposed protocol modifications to increase PTP robustness [17], including giving slaves the ability to check the calculated offset against time information from multiple NTP servers [18].

Another area where NTP has an advantage over PTP is security. NTP supports authentication with symmetric keys or public/private certificate pairs to allow clients to verify the authenticity and integrity of received messages. The standard IEEE 1588-2008, on the other hand, does not include any fully defined security model [19]. Security was not a priority in the development of PTP due to the typical use case under consideration at the time, i.e. time synchronization in closed local area networks (LANs) [20]. This means that in PTP, it is not possible to verify the authenticity and integrity of the critical *Announce* and *Sync* messages. This allows a malicious actor to influence the BMCA or time synchronization. Security researchers have shown that it is possible to cause a major disturbance in PTP synchronization via an *Announce* message Denial of Service attack on a slave. They were even capable of taking control of the whole PTP domain by creating an evil grandmaster, that claims better quality than other alternatives [21].

What PTP lacks in terms of robustness and security, it makes up for in accuracy. Typical accuracy expectations of PTP are in the order of 100 ns [22] while the typical values for NTP accuracy over the Internet range from 5 ms to 100 ms [23] if there is considerable delay asymmetry, such as when one direction is via satellite and the other via broadband.

One might ask why PTP accuracy and NTP accuracy differ so much when the protocols use an almost identical message exchange to calculate the clock offset. The difference in typically achieved synchronization accuracy has its origin in the vastly different networking environments the two protocols are used in.

PTP is primarily used in lightly loaded high-speed LANs. In these networks, overhead is of little concern, and update intervals of a few seconds or less can be used. Clocks lose their synchronicity over time because of changes in the physical environment (primarily temperature and barometric pressure) that affect the oscillator [24]. High-frequency update intervals allow clocks to re-synchronize faster. NTP requires long update intervals of one minute to several hours to minimize load on the typically heavily used network [22] [24]. NTP also operates in wide area networks (WAN), where differences in network speeds and routing paths are common sources of delay asymmetry. Furthermore, a significant amount of delay asymmetry can be removed from a PTP network by using only clocks that support hardware timestamping and connecting them exclusively via TCs or BCs. While hardware timestamping in clients and servers is rare but possible, NTP supports no mechanism for removing variations in queuing time in switches and routers [25].

Theoretically, a new version of NTP that uses the same delay asymmetry reduction strategies as PTP could be developed. If this were done, NTP could reach the same levels of accuracy and precision as PTP [22]. However, the current research focus is on improving PTP, rather than developing a more precise NTP.

IEEE 1588-2019 (PTP Version 2.1) is currently in the works. This new version addresses some of the robustness and security issues of PTP by enabling message and source integrity checking [26]. The next protocol version also allows sub-nanosecond accuracy and picoseconds precision of synchronization by incorporating the White Rabbit extension [27], which was developed at CERN, into the standard as a new configuration profile [28]. IEEE 1588-2019 is likely going to be released in early 2020 [29].

4. Conclusion

Choosing PTP as the time synchronization protocol for the important TSN effort, established PTP as the most important protocol for synchronizing clocks in real-time networks. PTP achieves high accuracy not by a novel way of calculating the offset of a clock, but through hardware timestamping and the usage of specialized network infrastructure devices. PTP currently lags behind NTP in the areas of robustness and security. Substantial changes to the protocol are needed to improve the protocol in these areas. It will be interesting to see if the next version IEEE 1588-2019 makes it possible to get both accuracy and security at the same time.

References

[1] K. G. Shin and P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, Jan 1994.

[2] IEEE 802.1 TSN Task Group. [Online]. Available: https://1.ieee802.org/tsn/

[3] A. Weder, "Whitepaper: Time Sensitive Networking," Tech. Rep. [Online]. Available: https://www.ipms.fraunhofer.de/de/press-media/whitepaper-download/TIME-SENSITIVE-NETWORKING-An-Introduction-to-TSN.html

[4] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–300, July 2008.

[5] "Whitepaper: Time Sensitive Networking," Tech. Rep. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf

[6] J. Eidson, *Measurement, Control, and Communication Using IEEE 1588*, ser. Advances in Industrial Control. Springer London, 2006.

[7] End-to-End Versus Peer-to-Peer. [Online]. Available: https://blog.meinbergglobal.com/2013/09/19/end-end-versus-peer-peer/

[8] The IEEE 1588 Default Profile. [Online]. Available: https://blog.meinbergglobal.com/2014/01/09/ieee-1588-default-profile/

[9] One-step or Two-step? [Online]. Available: https://blog.meinbergglobal.com/2013/10/28/one-step-two-step/

[10] PTP's Secret Weapon: Hardware Timestamping. [Online]. Available: https://www.corvil.com/blog/2016/ptp-s-secret-weapon-hardware-timestamping

[11] Protocols/ptp - The Wireshark Wiki. [Online]. Available: https://wiki.wireshark.org/Protocols/ptp

[12] What Makes a Master the Best? [Online]. Available: https://blog.meinbergglobal.com/2013/11/14/makes-master-best/

[13] NTP - The Wireshark Wiki. [Online]. Available: https://wiki.wireshark.org/NTP

[14] Sun Blueprint: Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP. [Online]. Available: http://www-it.desy.de/common/documentation/cd-docs/sun/blueprints/0701/NTP.pdf

[15] Combining PTP with NTP to Get the Best of Both Worlds. [Online]. Available: https://www.redhat.com/en/blog/combining-ptp-ntp-get-best-both-worlds

[16] P. V. Estrela and L. Bonebakker, "Challenges deploying PTPv2 in a global financial company," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, Sep. 2012, pp. 1–6.

[17] M. Dalmas, H. Rachadel, G. Silvano, and C. Dutra, "Improving PTP robustness to the byzantine failure," in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, Oct 2015, pp. 111–114.

[18] P. V. Estrela, S. Neusüß, and W. Owczarek, "Using a multi-source NTP watchdog to increase the robustness of PTPv2 in financial industry networks," in *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, Sep. 2014, pp. 87–92.

[19] RFC 7384 - Security Requirements of Time Protocols in Packet Switched Networks. [Online]. Available: https://tools.ietf.org/html/rfc7384

[20] K. O'Donoghue, D. Sibold, and S. Fries, "New security mechanisms for network time synchronization protocols," in *2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, Aug 2017, pp. 1–6.

[21] C. DeCusatis, R. M. Lynch, W. Kluge, J. Houston, P. Wojciak, and S. Guendert, "Impact of Cyberattacks on Precision Time Protocol," *IEEE Transactions on Instrumentation and Measurement*, pp. 1–1, 2019.

[22] IEEE 1588 Precision Time Protocol (PTP). [Online]. Available: https://www.eecis.udel.edu/~mills/ptp.html

[23] How does it work? [Online]. Available: http://www.ntp.org/ntpfaq/NTP-s-algo.htm

[24] D. Mills, *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*. CRC Press, 2017.

[25] NTP vs PTP: Network Timing Smackdown! [Online]. Available: https://blog.meinbergglobal.com/2013/11/22/ntp-vs-ptp-network-timing-smackdown/

[26] What's coming In the Next Edition of IEEE 1588? [Online]. Available: https://blog.meinbergglobal.com/2017/09/24/whats-coming-next-edition-ieee-1588/

[27] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez, "White rabbit: a PTP application for robust sub-nanosecond synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Sep. 2011, pp. 25–30.

[28] White Rabbit Official CERN website. [Online]. Available: http://white-rabbit.web.cern.ch/Default.htm

[29] iMeet Central. [Online]. Available: https://ieee-sa.imeetcentral.com/1588public/

An Overview on Vehicular Communication Standards

Kilian Zieglowski, Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: kilian.zieglowski@tum.de, kinkelin@net.in.tum.de

Abstract—**Communication between vehicles is evolving. It can be used for reducing dangerous situations, improving safety in traffic as well as making driving more convenient by enabling improved in-car entertainment such as Wifi and Video streaming. The main advantage is safer traffic with the goal of reducing traffic collisions to almost zero.**

There are two main techniques and standards that compete in the European market. These are the *C-ITS (Cooperative Intelligent Transport System)* of the EU based on the IEEE 802.11p standard, and the new developing cellular techniques using LTE and in the future the 5G radio standard. This paper shows an overview of the C-ITS and the cellular-based *ITS (Intelligent Transport System)* and its current standardised security architecture. C-ITS is mature and tested, whereas cellular ITS has multiple possible advantages for the future. However, cellular-based ITS still needs to be standardised and tested for vehicle communication.

Index Terms—**V2X, V2V, C-ITS, IEEE 802.11p, cellular ITS, ITS Security**

1. Introduction

The *IoT (Internet of Things)* is becoming an important part of our lives. More and more things are connected and becoming smarter. Meanwhile, IoT is even used in the automotive industry with the aim to make our transportation safer and more comfortable. This means making cars smarter and enabling them to communicate with each other in real-time. [1]

The new technology is called *V2X (Vehicle to Everything)* communication which is the summary of four different communication types [2]. These types are *V2V (Vehicle to Vehicle)*, *V2I (Vehicle to Infrastructure)*, *V2P (Vehicle to Pedestrian)* as well as *V2N (Vehicle to Network)* [2]. The V2X *ITS (Intelligent Transport System)* has four primary purposes [3]. First, ITS can be used for improving the safety of transportation with, e.g. assistance and warnings [4]. The primary purpose is to react on the not line of sight area, where the current sensors of a car are nowadays useless [5]. Second, ITS can be utilised to optimise traffic flow, e.g. platooning, which is the ability to link vehicles together in an automated way. [4]. Third, it can also be used for better in-car entertainment for business or pleasure by offering video-streaming or Wifi for mobile devices [6]. Fourth, autonomous driving can be improved with V2X, so a vehicle knows where the others are located, what they see and has the ability to predict the next most likely situation [3].

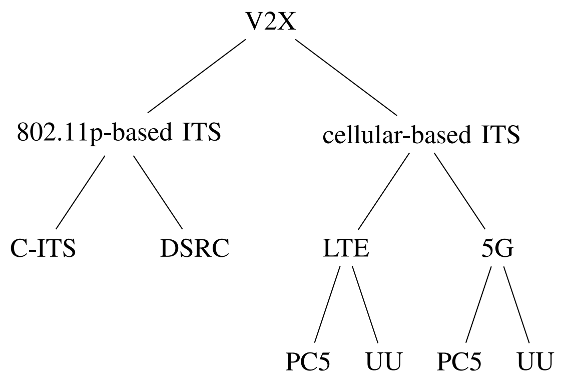


Figure 1: Taxonomy of used V2X communications

The V2X communication can be split into three leading technologies divided into two fields, which can be seen in figure 1. One field is the 802.11p-based standard, with the two technologies *DSRC (Dedicated Short Range Communication)* of the USA and the *C-ITS (Cooperative Intelligent Transport System)* of the EU [4]. Another field and technology is the cellular-based system. It uses for short direct communication the PC5 and for far infrastructure communication the UU interface, in LTE and the upcoming 5G standard [4].

The aim of this paper is to give an overview which will include advantages and disadvantages of the IEEE 802.11p-based communication, with a special focus on C-ITS in Section 2, which will be followed by LTE/5G-based communication in Section 3. Furthermore, opportunities and difficulties of the coexistence of LTE-based and IEEE 802.11p-based ITS are detailed in Section 4. In Section 5, the security is described with the features of IEEE 802.11p-based ITS in Subsection 5.1 as well as of the LTE-based ITS in Subsection 5.2. In Section 6, some related work is listed, which is followed by the conclusion in Section 7.

2. IEEE 802.11p-based ITS

IEEE 802.11p is based on the normal Wifi standard IEEE 802.11a, which is broadly used in private Wifi environments and is adapted for the use in V2X communication [5]. The V2V communication based on the Wifi Standard 802.11p standard is mature and well tested, which makes it a technique ready to use [7]. The 802.11p is divided into two primary standards: the DSRC of the USA and the C-ITS of the EU [4].

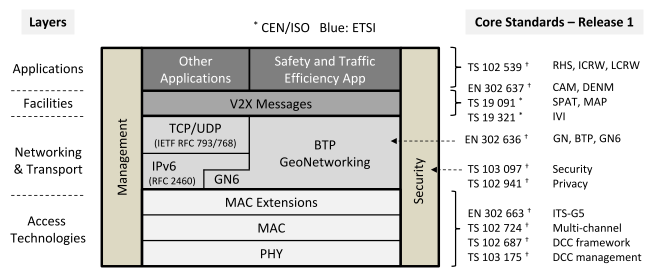


Figure 2: C-ITS protocol stack and standards [4]

In the following, the focus is set on the C-ITS of the EU, because of multiple similarities between DSRC and C-ITS. The idea of the C-ITS was to spread information about, e.g. speed and direction to vehicles nearby and if necessary use multi-hop to reach afar vehicles [4]. Moreover, the EU wants to develop the ITS system without any infrastructure with the main focus on V2V communication [8]. It can be seen as a network of vehicles communicating with each other and relaying messages. This network decreases the expenses of each car owner by excluding costly infrastructure. However, for effective usage, a minimum of 10% of all the cars need to be equipped with the ITS system to have a noticeable impact on safety [8]. This equipment quantity would be reached in 2.5 years if every second newly released vehicle would be equipped with C-ITS [8].

C-ITS operates like DSRC in the 5 GHz frequency band and uses the same technology in the lower PHY and MAC protocol layer [4], which can be seen in figure 2. The single-hop communication ranges from 10 metres up to 1 kilometre depending on the weather conditions and if it is in line of vision [4]. The communication can be further extended by multi-hop message transport [4]. The multi-hop communication uses geographical data for effective routing [4]. This routing is needed because not every vehicle in range should resend each message [4]. Such a resending of every vehicle in proximity could lead to an overhead and breakdown of the system [4]. Therefore, the selection of vehicles utilises an algorithm, which uses the broadcast information from the vehicles about their location and their neighbours [4].

Furthermore, the channel width is reduced to 10Mhz, as a result of robustness issues [8]. That limits the communication data rates to 27 Mb/s, which can be reduced to 3 Mb/s to react to interferences and enable a larger communication range and a lower packet failure rate [8]. Nevertheless, this low data rate limits the usability in the entertainment segment, e.g. video streaming of the infrastructure or gaming between vehicles [8]. Moreover, 802.11p has a high potential of errors in high-density vehicle conditions, no exact future enhancement plans, or usable and buildable *RSUs (Road Side Units)* [7]. However, for safety-relevant communication it is crucial to support a low latency real-time communication and C-ITS enables typical end to end latencies of under 10 ms [9].

802.11p is adapted to the high mobility in vehicle communication, with a maximum operating speed of 500 km/h by handling doppler effects and frequent changing multi-path reflections [5]. Notwithstanding, there are scalability issues in high-density areas such as traffic jams because just a single device is capable of sending a package in a certain time and channel in DSRC [10].

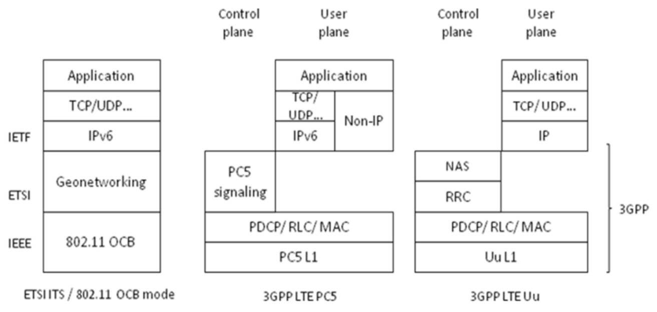


Figure 3: Protocol stack and standards for LTE-UU and PC5 in comparison to C-ITS [13]

3. Cellular-based ITS

The use of LTE for V2X communication is still immature and so far not ready to use, but it has multiple advantages today and in future [7]. One aspect is the steady development and improvement of specifications and capabilities of the general cellular standards, which are the basis for cellular V2X communication [7]. The main advantage of cellular-based communication is the already existing infrastructure [4] with a global deployment [3]. The quality of service is guaranteed by cellular providers, which offer and enlarge their infrastructure for other use cases such as smartphones. Nearly every new vehicle uses the cellular network and has already the infrastructure on board for, e.g. traffic information [11]. Most manufacturers offer information systems such as *RTTI (Real Time Traffic Information)* which provide data about, e.g. road usage, defect cars or tailbacks [11] for safety and non-safety relevant information.

By developing LTE and *LTE-A (LTE Advanced)*, it evolved to be a worthy competitor for the 802.11p. LTE-A supports a mobility speed of up to 350 km/h, a maximum data rate of 1 Gb/s and a range of up to 30 kilometres [12] and for the first time a direct communication between devices [3].

V2I/V2N communication is realised by the LTE-UU interface [3]. By enabling *D2D (Device to Device)* in LTE-A [3], a direct V2V communication is possible on the basis of a PC5 interface [14]. In LTE Advanced Release 12 of 3GPP, LTE-Direct was supported for the first time and made direct communication between devices in proximity possible [15]. The cooperation 3GPP standardises different standards for LTE-UU and PC5 such as the ETSI does for the C-ITS a comparison can be seen in figure 3. The authentication and timing are initialised via the infrastructure [15]. After that, the devices can communicate directly [15]. The D2D communication is also possible if there is no base station in range or if it is damaged [16]. This extends the working area and saves battery power [16]. However, the D2D system was still insufficient and unspecialised for vehicle speed, so in Release 14 of 3GPP it got adapted to the requirements of V2V communication [16]. Furthermore, the cellular V2X system is capable of integrating other entities such as pedestrians which enables V2P communication in their system by using the PC5 interface and LTE smartphones [17]. The integration would be beneficial, especially for urban areas with the aim of smart cities [17].

By the use of 5G and its adaption for the communication between machines, the latency of 10 ms to 100 ms of previous generations decreases to 1 ms, as a result of complex back coupling [18]. 5G also enables peak data rates faster than 10 Gb/s, and it supports more than 1 million devices per square kilometre [15]. Moreover, 5G will support a maximum mobility speed of 500 km/h [19]. That means it would also be possible to transport security-relevant information using the new cellular standard.

In future, the LTE module in cars can be replaced with an LTE module which enables both V2V by PC5 and V2I by LTE-UU with only one Chip [17]. The LTE network can handle a high density of devices since it is capable of frequency-domain multiplexing of many devices [10].

4. Coexistence of LTE-based and IEEE 802.11p-based ITS

802.11p, as well as the LTE PC5 standard, are operating on the 5 GHz simultaneously, which would lead to interferences [6]. The *5G Automotive Association (5GAA)*, which support the cellular-based standard, and *car to car communication consortium (C2C-CC)*, which support the WIFI-based communication, are in contact for developing a method so that both technologies can coexist without interferences or malfunctions [15]. 5GAA made a proposal to separate the frequencies among them, which has been rejected by the C2C-CC because the proposed frequencies are used by them [15]. Also, it is crucial to be prepared for prospective challenges and not to rule out one system [15]. Nevertheless, the coexistence of the two systems would be more expensive than just integrating and developing a single one [17].

5. Security

Security is essential for every ITS system because it is processing critical data. Moreover, it is important that every authentication method is able to exclude devices out of the network, which are sending wrong data.

Currently, there are just a few cars which allow direct V2V communication [15] like the new VW Golf 8 [20]. Consumer acceptance is a crucial point that needs to be considered for the integration of ITS. Therefore, it needs to comply with safety and privacy requirements. For secure V2X communication, it is necessary that authentication, authorisation, availability, data confidentiality and data integrity need to be fulfilled to be prepared for all possible types of attack [3]. Finally, it is much needed for privacy reasons to anonymise the data communication so that personal data is secure, and users can not be traced [3].

5.1. IEEE 802.11p-based ITS

In C-ITS the authentication can mainly be separated into two parts, the direct V2V communication and V2I communication. DSRC uses *WAVE (Wireless Access in Vehicular Environments)* which is the combination of IEEE 802.11 and IEEE 1609 standard [4]. IEEE1609.2 defines an authentication method which is useful for V2I [21]. Furthermore, it provides secure communication standards with the use of *PKI (public key infrastructure)* [22]. A *CA (certificate authority)* validates identities and signs certificates [21]. Those certificates can be transferred via DSRC as well as be used for the authentication of messages [21].

In the EU, the security standard for V2X communication is standardised by the ETSI [21]. In the C-ITS, secure access is handled similarly to DSRC and is just broader because of the used GeoNetworking for multi-hop messages [21]. The C-ITS uses as network protocol IPv6 in combination with UDP and TCP as transport protocol [4].

Certificates are also used to support security and privacy [23]. PKI can be used for signing messages, as well with pseudonyms for privacy reasons, to enable secure communication for V2X [23]. The Root-CA is superordinated, which gives certificates to every sub-CA or *EE (End Entity)* [23]. There are at least two sub-CAs: the *EA (Enrollment Authority)* and *AA (Authorisation Authority)* [23]. Each vehicle has a personal signature and a static public key for initialisation whereby the vehicle gets an *EC (Enrollment Credential)* certificate of the EA, which is valid for a few years [23]. An EC is updated shortly before it expires using the still valid EC at EA for verification [23]. Furthermore, the EC of the EA can be used to get short-living *ATs (Authorisation Tickets)* of the AA for the usage between vehicles and infrastructure [23]. The vehicle sends a request to the AA as well as the EC to the EA [23]. If the EC is valid, the EA sends the confirmation to the AA, which gives ATs to the vehicle [23]. For safety reasons, a *CRL (Certificate Revolution List)* is used for revoking access to the system for possible malicious entities, where the ECs are listed [23]. If the EC is not valid, the certificate is sent to the CRL [23]. The separation of authentification of EE in EA and the certificate issuance of AA to EE is for anonymising the exact identity of users, so they can not be traced [23]. The EA saves data of the vehicles for identification on which the AA has no access to [23].

In future, there will be a certificate policy with some requirements for the certificates, but every CA can add requirements for its use [23]. There will be multiple CAs in Europe, and for coordinating certificates between all European countries *TLM (Trust List Manager)* will be used [23]. The TLM lists all certificates so that vehicles can be validated in other states and CAs [23]. The TLM list itself is secured by a certificate with a public key which has to be sent to every entity to get access to the list of all valid certificates in order to communicate with each other [23].

The certificate in a *CAM (Cooperative Awareness Message)* can be reduced to an 8-bit hash code for reducing the load on the communication channel [23]. Furthermore, the hash code and the certificate are saved by the receiver and is updated only once every second [23]. When receiving a message, the hash code is used for verification [23]. CAMs are sent periodically and frequently [23]. The data of CAMs, e.g. speed, position and steering of the own and neighbours vehicles are exchanged regularly and collected in a local dynamic map [23].

GeoNetworking uses the information about the positions of vehicles for efficient routing [4]. Besides, it enables an exchange of information in a specific geographical area in order not to be restricted by the signal range of a single-vehicle [4]. For transmitting IPv6 packets using GeoNetworking, a sublayer *GN6 (IPv6 over GeoNetworking)* was standardised [4].

A *DENM (Decentralised Environmental Notification Message)* can make multi-hop and is sent in potentially dangerous situations [23]. Those Situations may be, e.g. obstacles on roads, adverse weather conditions or road works [23]. DENMs can be resent and updated in case of longer-lasting dangerous activities and are cancelled when there is no more danger [23]. DENMs are event-driven and are sent when something happens and are not sent regularly in comparison to CAMs [23].

The lower protocol layers PHY and MAC are not expected to be modified in the future so that the suboptimal performance may stay the same [4]. However, it will be modified in the upper layers by superior algorithms for spreading information, with an increase in safety and performance [4].

ECDSA (Elliptical Curve Digital Signature Algorithm) [24] is specified as the signature algorithm, with the usage of the NIST P256, which is an elliptical curve [23]. In future, different algorithms are needed to provide security like the Brainpool curve with 256 or 384-bit length [23]. These are standardised in the ETSI standard TS 103 097 [23].

5.2. Cellular-based ITS

LTE has some standard security mechanisms, but they are not sufficient in a V2V direct communication using the PC5 interface without using the base station [16]. Therefore, the security mechanisms can be divided into two main parts, the direct communication via the PC5 interface and the communication with the infrastructure via the UU interface, and a future enhancement in 5G.

The LTE system can be used for V2N communication. However, vehicles have to be authenticated by the infrastructure and authorised for V2X communication as well as the vehicle has to authenticate the infrastructure [3]. For secure communication of V2X service, the *LTE-AKA (LTE Authentication and Key Agreement)* protocol is used, which is provided by the LTE security framework [3]. LTE-AKA protocol is used for identification, authorisation and key sharing and derivation for facilitating secure wireless access [3]. Nevertheless, the present LTE-AKA is not adapted to, e.g. the high mobility of V2I communication which leads to longer transmission and end-to-end latency times [3]. Additionally, in LTE-AKA the specific identity of the entity is not hidden [3]. Hence, the entities are traceable, and it is possible that a malicious entity uses a man-in-the-middle attack between the vehicle and the infrastructure by using his identity and sending wrong data [3]. This can lead to misinterpretation of vehicles and can result in crashes [3]. Therefore the LTE-AKA protocol has to be further extended for safer, faster and anonymous communication.

Similar to V2I communication, mutual authentication between the vehicles is needed to avoid malicious spread of information [3]. For V2V authentication of communication in the PC5 interface, the ProSe security framework can be used [3]. Furthermore, the ProSe D2D of 3GPP communication can be utilised for authorisation, authentication and discovery [3]. Some differences are that vehicles do not have electricity or computing capacity issues like other mobile devices, what ProSe was initially developed for, but they have a high mobility [3]. Hence, a secure access and communication method specialised for vehicles still need to be standardised [17].

In comparison to that, 5G has multiple use cases and therefore needs a flexible authentication method which supports the variety of requirements [25]. The 5G frequency will support an optimised direct communication for vehicles, where the special security mechanisms of the LTE D2D can be extended [3]. The secure access can be divided into two securing ranges. The first mandatory authentication is for general access to the core 5G [3]. The second authentication will be optional, which is based on protocol configuration options, where the PAP/CHAP user credentials are listed [3].

There are a lot of different techniques to enable secure communication and authentication. Some mechanisms would use the *DHKE (Diffie-Hellman Key Exchange)* generation of symmetric keys or shared keys for a V2I communication [2]. Lastly, TLS or SSL can be used for secure communication, but a distinct disadvantage is that the identity of the entity is visible, which leads to privacy issues [2].

6. Related Work

Beside the C-ITS system of the EU and the cellular-based ITS system, which are explained in this paper, some more systems for ITS are in the development phase. Some other nations made their own tests on ITS such as Australia, Japan, China or South Korea [15]. Furthermore, there is another technology tested for ITS, the *WiMAX (Worldwide Interoperability in Microwave Access)*, but with an insignificant role in the market [3] [26]. Moreover, other countries have their own system and use other frequencies like Japan, which develops an ITS system similar to DSRC of the US [27] [8]. Some other technologies, for example, visible light communication or mmWAVE, were also tested for ITS [15].

7. Conclusion

Both the C-ITS and the cellular-based ITS have advantages and disadvantages. The C-ITS is a mature technology which is ready to use. It offers direct V2V communication as well as V2I, which has small latencies. Furthermore, it allows for more extensive communication ranges the use of equipped vehicles in the middle as a repeater. However, up to now, there is no existing infrastructure on which the system could build on. Moreover, there are no RSUs which are ready to build. Nevertheless, this technology is ready to be published and spread and could save our lives soon.

On the other side, the cellular-based ITS uses the already existing infrastructure which is used for, e.g. smartphones and enables a faster impact on safety, because of this infrastructure. Additionally, it provides the new PC5 interface, which allows a direct device communication and therefore is a strong competitor for C-ITS. Besides allowing standard cellular communication by using an infrastructure, it also offers a V2V or V2P direct communication. This direct communication reduces the latency time, which was a significant disadvantage for cellular-based ITS. Furthermore, it enables communication where no infrastructure is needed, and it enables communication in areas with insufficient cellular coverage or outdated technology standards, which would have large latencies. However, the cellular-based standard for ITS is not fully standardised and not ready to use, e.g. it has some uncertainties in the secure access control, which will take some time for advancement.

The integration of pedestrians over smartphones is a substantial benefit of cellular-based communication, which would enable the integration of vulnerable roadside users such as pedestrians or cyclists. Beyond that, the ability to use smartphones for communicating with vehicles enables the integration of old cars. The C-ITS would need RSUs for communication with other devices, whose infrastructure was not initially planned by the EU.

References

[1] H. Holland, *Connected Cars*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 51–81. [Online]. Available: https://doi.org/10.1007/978-3-658-22929-0_3

[2] K. J. Ahmed and M. J. Lee, "Secure LTE-Based V2X Service," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3724–3732, Oct 2018.

[3] M. Muhammad and G. A. Safdar, "Survey on existing authentication issues for cellular-assisted V2X communication," *Vehicular Communications*, vol. 12, pp. 50 – 65, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209617302267

[4] A. Festag, "Standards for vehicular communication—from IEEE 802.11p to 5G," *e & i Elektrotechnik und Informationstechnik*, vol. 132, no. 7, pp. 409–416, Nov 2015. [Online]. Available: https://doi.org/10.1007/s00502-015-0343-0

[5] A. Filippi, K. Moerman, V. Martinez, A. Turley, O. Haran, and R. Toledano, "IEEE802. 11p ahead of LTE-V2V for safety applications," *Autotalks NXP*, 2017.

[6] N. Xia and C.-S. Yang, "Vehicular Communications: Standards and Challenges," 2017.

[7] A. Bazzi, B. M. Masini, A. Zanella, and I. Thibault, "On the Performance of IEEE 802.11p and LTE-V2V for the Cooperative Awareness of Connected Vehicles," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 10 419–10 432, Nov 2017.

[8] R. K. Schmidt, T. Leinmüller, and B. Böddeker, "V2x kommunikation," in *In Proceedings of 17th Aachener Kolloquium*, 2008.

[9] C. Ress and M. Wiecker, "Potenzial der V2X-Kommunikation für Verkehrssicherheit und Effizienz," *ATZ - Automobiltechnische Zeitschrift*, vol. 118, no. 1, pp. 16–21, Jan 2016. [Online]. Available: https://doi.org/10.1007/s35148-015-0154-y

[10] H. Seo, K. Lee, S. Yasukawa, Y. Peng, and P. Sartori, "LTE evolution for vehicle-to-everything services," *IEEE Communications Magazine*, vol. 54, no. 6, pp. 22–28, June 2016.

[11] (2019). [Online]. Available: https://www.bmw-me.com/en/topics/fascination-bmw/connected-drive/rtti.html

[12] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro, "LTE for vehicular networking: a survey," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 148–157, May 2013.

[13] T. Yoshizawa and B. Preneel, "Survey of security aspect of v2x standards and related issues," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2019, pp. 1–5.

[14] Y.-L. Tseng, "LTE-advanced enhancement for vehicular communication," *IEEE Wireless Communications*, vol. 22, no. 6, pp. 4–7, 2015.

[15] B. Masini, A. Bazzi, and A. Zanella, "A survey on the roadmap to mandate on board connectivity and enable V2V-based vehicular sensor networks," *Sensors*, vol. 18, no. 7, p. 2207, 2018.

[16] V. Marojevic, "C-V2X Security Requirements and Procedures: Survey and Research Directions," 2018.

[17] T. Rebbeck, J. Stewart, H.-A. Lacour, A. Lillen, D. McClure, and A. Dunoyer, "Socio-economic benefits of cellular V2X," *Final Report for 5GAA.[(accessed on 31 August 2019)]*, 2017.

[18] R. Freund, T. Haustein, M. Kasparick, K. Mahler, J. Schulz-Zander, L. Thiele, T. Wiegand, and R. Weiler, *5G-Datentransport mit Höchstgeschwindigkeit*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 89–111. [Online]. Available: https://doi.org/10.1007/978-3-662-55890-4_7

[19] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The next generation wireless access technology*. Academic Press, 2018.

[20] (2020). [Online]. Available: https://www.volkswagen.de/de/modelle-und-konfigurator/der-neue-golf.html#iqdrive

[21] A. Weimerskirch, "V2X security & privacy: the current state and its future," in *ITS World Congress, Orlando, FL*, 2011.

[22] A. Rao, A. Sangwan, A. A. Kherani, A. Varghese, B. Bellur, and R. Shorey, "Secure V2V Communication With Certificate Revocations," in *2007 Mobile Networking for Vehicular Environments*, May 2007, pp. 127–132.

[23] T. Strubbe, N. Thenée, and C. Wieschebrink, "IT-Sicherheit in Kooperativen Intelligenten Verkehrssystemen," *Datenschutz und Datensicherheit - DuD*, vol. 41, no. 4, pp. 223–226, Apr 2017. [Online]. Available: https://doi.org/10.1007/s11623-017-0762-7

[24] B. Brecht and T. Hehn, *A Security Credential Management System for V2X Communications*. Cham: Springer International Publishing, 2019, pp. 83–115. [Online]. Available: https://doi.org/10.1007/978-3-319-94785-3_4

[25] X. Zhang, A. Kunz, and S. Schröder, "Overview of 5G security in 3GPP," in *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, Sep. 2017, pp. 181–186.

[26] M. S. Anwer and C. Guy, "A survey of VANET technologies," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 5, no. 9, pp. 661–671, 2014.

[27] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions," *IEEE Communications Surveys Tutorials*, vol. 13, no. 4, pp. 584–616, Fourth 2011.

ISBN 978-3-937201-69-6
9 783937 201696

ISBN 978-3-937201-69-6
DOI 10.2313/NET-2020-04-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)